

AD-A040 466

HARVARD UNIV CAMBRIDGE MASS AIKEN COMPUTATION LAB  
RESEARCH ON VIRTUAL MACHINES.(U)  
MAY 77 U O GAGLIARDI, R K JAIN

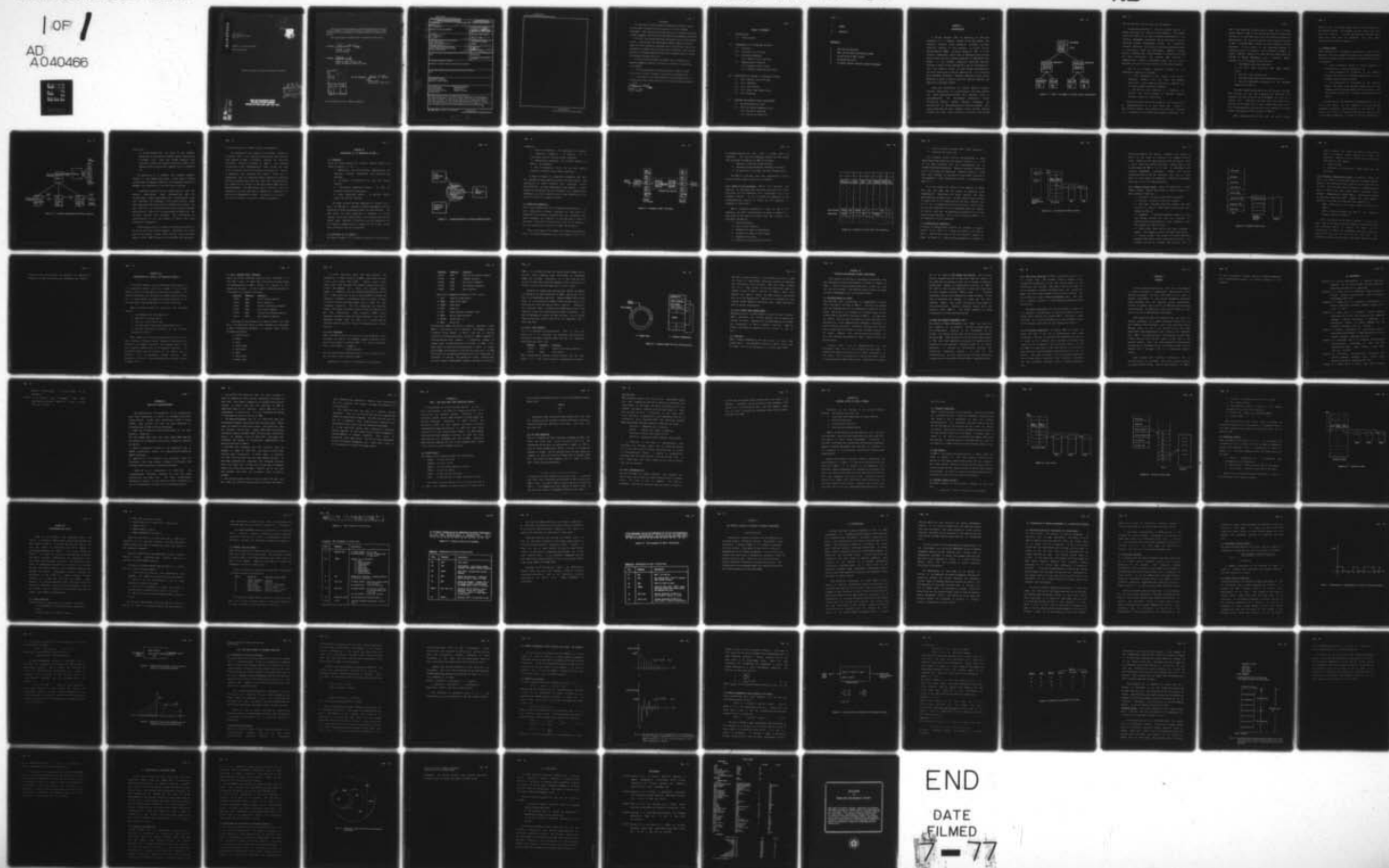
F/G 9/2

UNCLASSIFIED

RADC-TR-77-57

F30602-76-C-0001  
NL

1 OF 1  
AD  
A040466



END

DATE

FILMED

7-77

AD A 040466

RADC-TR-77-57  
Final Technical Report  
May 1977

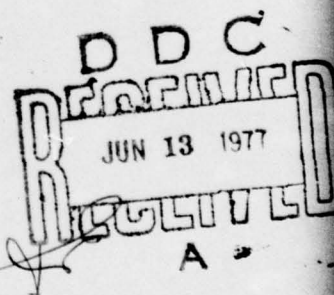
RESEARCH ON VIRTUAL MACHINES

Harvard University

12  
NW



Approved for public release; distribution unlimited.



AD No. \_\_\_\_\_  
DDC FILE COPY

ROME AIR DEVELOPMENT CENTER  
AIR FORCE SYSTEMS COMMAND  
GRIFFISS AIR FORCE BASE, NEW YORK 13441

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

This report has been reviewed and is approved for publication.

APPROVED: *Raymond A. Liuzzi*

RAYMOND A. LIUZZI  
Project Engineer

APPROVED: *Robert D. Krutz*

ROBERT D. KRUTZ, Colonel, USAF  
Chief, Information Sciences Division

APPROSSION FOR	
NTIS	White Sec. <input checked="" type="checkbox"/>
DOC	Buff Sec. <input type="checkbox"/>
UNANNOUNCED	<input type="checkbox"/>
JUSTIFICATION	
BY	
DISTRIBUTION/AVAILABILITY CODES	
Dist.	AVAIL. and/or SPECIAL
A	

FOR THE COMMANDER:

*John P. Huss*

JOHN P. HUSS  
Acting Chief, Plans Office

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-77-57 ✓	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) RESEARCH ON VIRTUAL MACHINES.	5. TYPE OF REPORT & PERIOD COVERED Final Technical Report, June 1975 — Oct 1976.	6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) Ugo O. Gagliardi R. K. Jain	8. CONTRACT OR GRANT NUMBER(s) F30602-76-C-0001 new	9. PERFORMING ORGANIZATION NAME AND ADDRESS Harvard University Aiken Computation Laboratory ✓ Cambridge MA 02138
10. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIM) Griffiss AFB NY 13441	11. REPORT DATE May 1977	12. NUMBER OF PAGES 89
13. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same	14. SECURITY CLASS. (of this report) UNCLASSIFIED	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A
16. DISTRIBUTION STATEMENT (of this Report)  Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Same		
18. SUPPLEMENTARY NOTES RADC Project Engineer: Raymond A. Liuzzi (ISIM)		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Virtual Machines                      Memory Management Operating Systems                      Peripherals Virtual Machine Monitor              Computer Software Virtualization		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A Virtual Machine Monitor for a PDP 11/40 mini-computer is being developed for research purposes. A Virtual I/O module, and a portion of the VMM Kernel have been developed. The VMM provides virtual I/O to several I/O devices like Disc, tape, printer, etc. via an X-Bus connected to a PDP/10. The VMM also provides multiple virtual machine structures so that these virtual devices can be divided among various virtual machines in any desired manner. ↑		

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

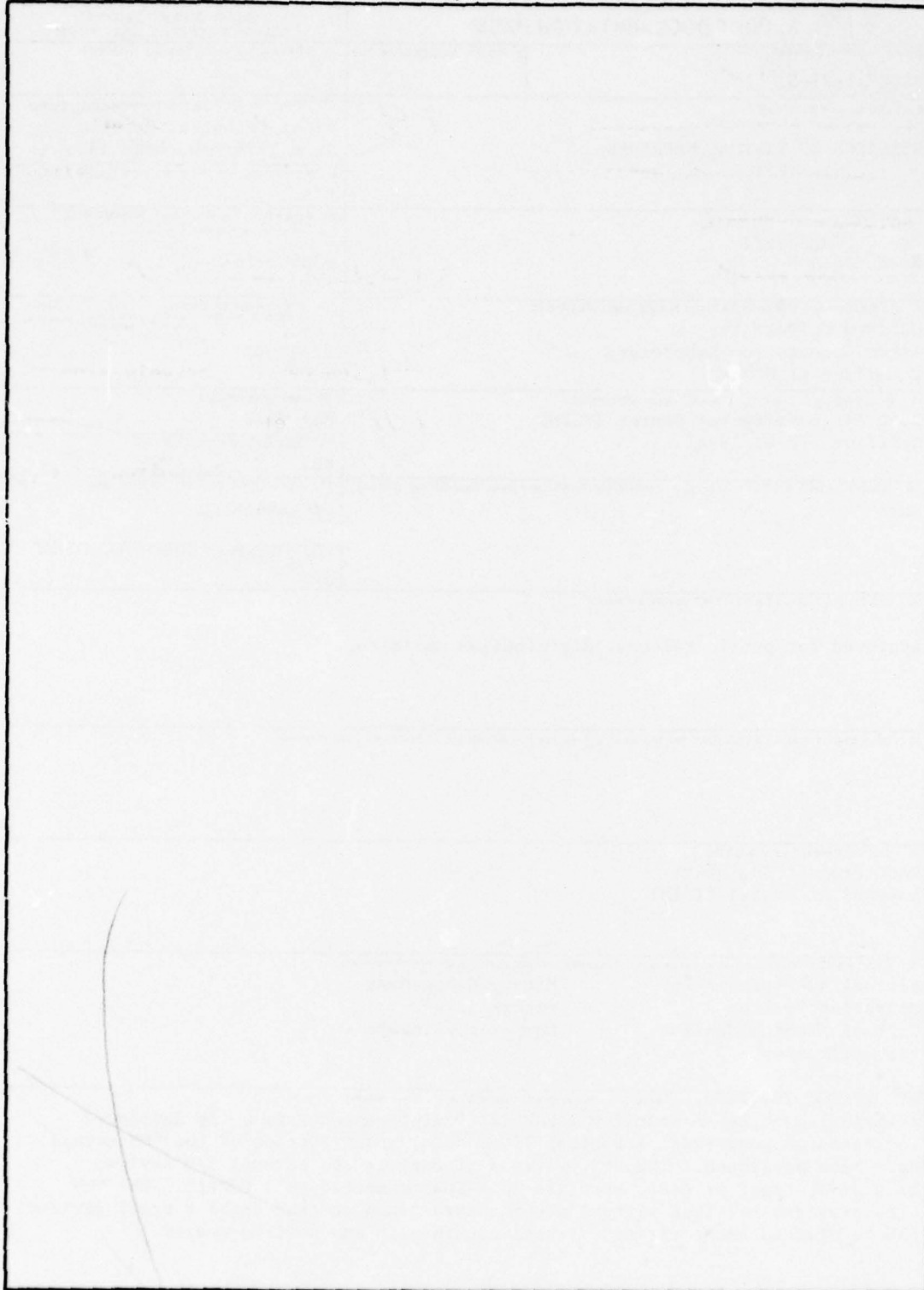
405831

Junc



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)



UNCLASSIFIED

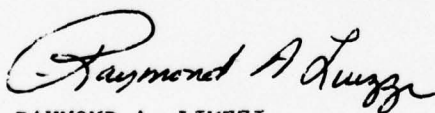
SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

## EVALUATION

The literature on Virtual Machine Technology has indicated a number of areas where virtualization can be useful as a tool for software development. These applications include areas such as, software reliability, software debugging, operating system enhancement and programmer productivity.

This effort has provided additional applications of the Virtual Machine concept. In particular, the Virtual Machine Monitor (VMM/11) developed under this effort provides an environment which can effectively aide in the development of software that is reliable and cost effective. This facility provides the following capabilities which make it ideally suited to explore new debugging techniques and tools:

- a. The PDP 11/40 environment and VMM/11 offer an opportunity for extending debugging techniques it utilizes on the PDP/10 to other hosts on the ARPANET.
- b. The Virtual Machine technology developed for VMM/11 provides a facility well suited to building a wide variety of debugging tools.
- c. The concept of utilizing peripherals of another system while in a virtual mode offers needed flexibility to the system software developer.



RAYMOND A. LIUZZI  
Project Engineer

## TABLE OF CONTENTS

- I. INTRODUCTION
  - 1.1 Previous Work
  
- II. TECHNIQUES OF I/O HANDLING IN VMM-11
  - 2.1 Overview
  - 2.2 Structure of I/O module
  - 2.3 Detection submodule
    - 2.3.1 Table of I/O locations
  - 2.4 Communication submodule
    - 2.4.1 Request Control Block
    - 2.4.1 Software Communication Words
  
- III. DESCRIPTION OF VIRTUAL I/O DEVICES IN VMM-11
  - 3.1 RK-11 DECpack Disk Cartridge
  - 3.2 TC-11 DECTapes
  - 3.3 LP-11 Line Printer
  - 3.4 PC-11 Paper Tape Reader/Punch
  - 3.5 LT-33 Teletype
  
- IV. PROBLEMS ENCOUNTERED DURING DEVELOPMENT
  - 4.1 Mulfunctioning of X-BUS
  - 4.2 Need for Memory Management Unit
    - 4.2.1 Bus Error Handling
    - 4.2.2 Interrupt Simulation

V. SUMMARY

VI. REFERENCES

APPENDICES

- A. PDP-11/40 Virtualization
- B. MUNI: The Mini UNIX like operating system
- C. Current Status of VMM-11 kernel
- D. Programming the X-BUS
- E. On Control Theoretic Approach to Memory Management



## CHAPTER I

### INTRODUCTION

A Virtual Machine (VM) is basically an efficient simulation of a computer system on the same system. The simulator software, which generally provides multiple identical replicas of host machine, is called Virtual Machine Manitor (VMM). A classic example of VMM is CP-67 (Control Program-67), which runs on IBM-360/67 and provides several exact replicas (Virtual Machines) of IBM-360/67 (see Figure 1.1) The software (generally Operating Systems) running on these virtual IBM-360/67's can not distinguish it from real machine except for some timing considerations. The exact definition of Virtual Machine and it's diference from Emulated Machines, Extended Machines, and/or Pseudo Machines is given in Buzen & Gagliardi [BuG73A, BuG73B], and Gagliardi & Goldberg [GaG72].

Since the introduction of Virutal Machine concept, numerous application of virtualization have been pointed out. These applications include such important areas as Instrumentation for Performance Evaluation, System Reliability Studies, System Software Debugging, and Introduction of New peripherals to existing systems. This report describes one such example, where Virtual Machine concept has been used to provide I/O devices, even though

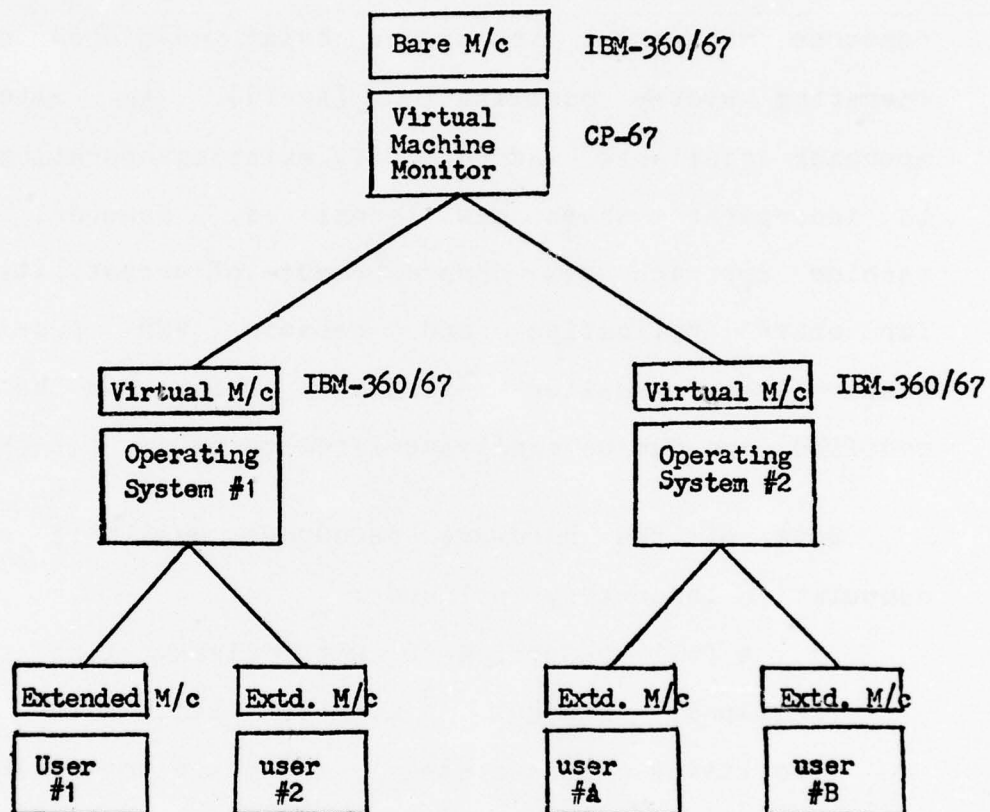


Figure 1.1 : CP-67 - An example of Virtual Machine Organization

the real machine does not have any peripherals.

At Harvard University, a virtual machine monitor is being developed for PDP-11/40 mini-computer. The primary aim of this VMM is to be able to measure performance of new resource management techniques being developed here for operating system optimization [Arn76]. An alternative approach available was to modify existing operating system to incorporate these new techniques. However, virtual machine approach was chosen because of versatility of VMM for other application and because VMM provides a comparatively smaller environment which can be easily modified, and can be easily verified to be correct [BuG74].

Some of the hardware resources available at our computation laboratory included :

1. A full DECsystem-10 with disks, line printer, DECTapes, several dial up lines and graphic facilities. This system is connected to ARPANET and is heavily used by graduate students.
2. Two PDP-11's each connected to a TELETYPE, but without any other hardware or software support. Obviously these are rarely used.

Because of heavy user load on PDP-10, any possibility of experimentation on it were ruled out. Instead a PDP-11/40 was chosen. The required I/O support was provided by a combination of hardware and software techniques. The

PDP-11 was connected to PDP-10 via an X-BUS and a Virtual Machine Monitor (VMM-11) was written for PDP-11 so that VM's running on it could make use of all the devices available on DECsystem-10 just as if the devices were connected directly to VM's. Although VMM-11 project is still underway, a first byproduct of the effort is the experience gained in providing virtual I/O support. The I/O module of VMM-11 is almost complete except for a small portion whose operation depends on Memory Management unit. Currently VMM-11 provides I/O to following virtual devices :

1. One DECpack Disk Cartridge RK-11.
2. One TC-11 Tape controller with eight DECTape units.
3. One LP-11 Line Printer unit.
4. One High Speed Paper Tape Reader/Punch PC-11.
5. One LT-33 TELETYPE (in addition to the TELETYPE connected to PDP-11).

The model numbers noted above are for virtual devices. Real devices can be, and in general are, different. For example, the real device for LT-33 TELETYPE I/O can be a display unit. Similarly, the real device for Line Printer may also be a display unit or even a paper tape punch unit. The I/O configuration and real-to-virtual mapping is defined before starting the PDP-10 portion of VMM-11.

VMM-11 supports multiple VM's and the above virtual



devices could be divided among various virtual machines in any desired fashion. For example, one VM could have Disk and Line Printer, a second VM could have DECTapes and a third one could have the TELETYPE, and High Speed Reader/Punch as shown in Figure 1.2 .

#### 1.1 PREVIOUS WORK :

Some techniques of introducing peripherals using Virtual Machines have been suggested by Buzen and Goldberg [BuG74]. They have pointed out the following 3 rationales for virtual I/O support:

1. A new peripheral device is being proposed or developed for an existing product line.
2. A new peripheral is introduced in the computer system and there is no software support for it in the commonly used operating system.
3. A new peripheral is introduced in the computer system and there is no software support for it in the commonly used operating system, but there exists some specialized stand-alone software which supports the device.

Frasson [Fre73] has described an implementation of the first case above. He has suggested a special purpose language to describe characteristics of new device. The method of Virtual I/O support described here does not fall in the above categories. Instead, it can be described as a

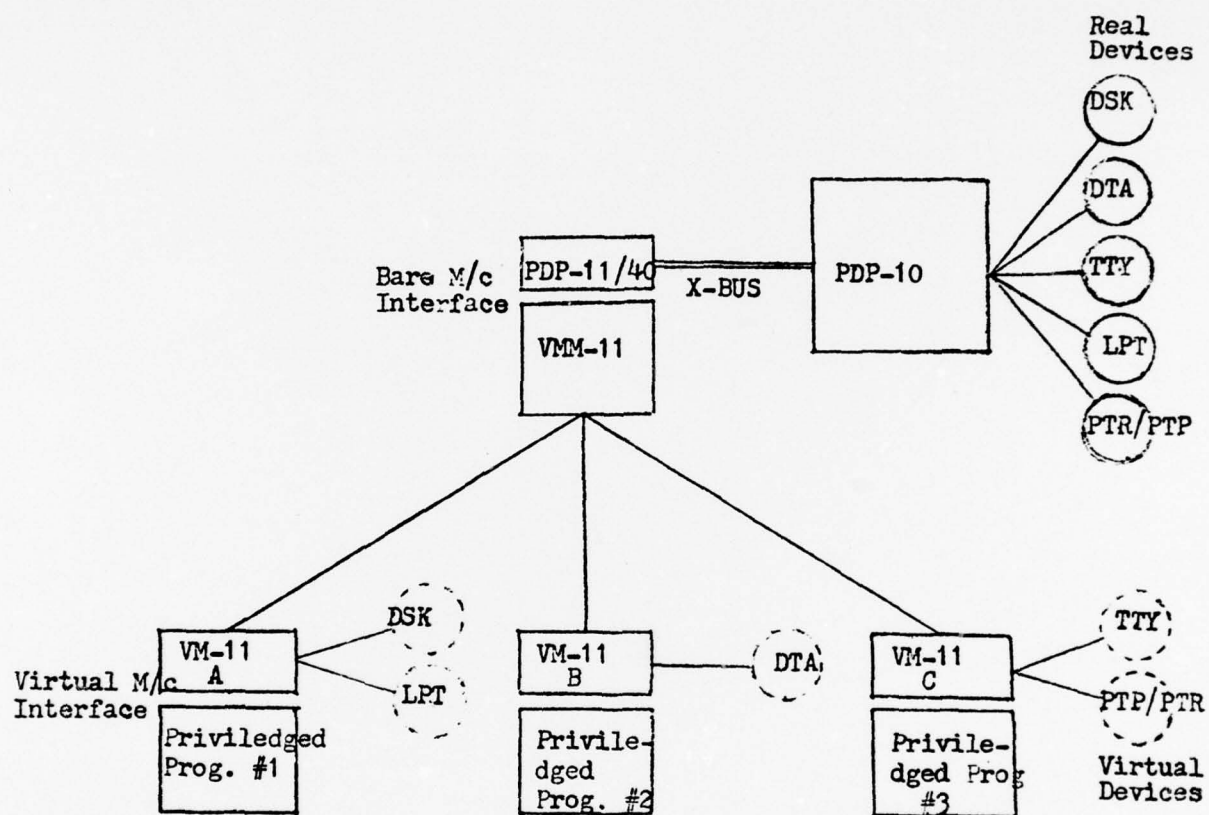


Figure 1.2 : A sample configuration of Virtual Machines.

fourth case :

4. The peripheral does not exist on the computer system but is available on another system connected by a hardware link. Thus even though commonly used operating system does support the device, there is no physical device on the host computer to do or simulate I/O.

The existence of a hardware link between computer systems is very common these days. In the light of recent rapid growth of computer networks, and tremendous success of ARPANET, the importance of this 4th case is obvious.

A necessary condition for all above cases is that the machine architecture under consideration must be virtualizable. Exact requirements for virtualization have been specified in Goldberg's Ph.D thesis [Gol72]. Using these specifications Popek & Kline [PoK76] have pointed out that PDP-11/45 is virtualizable only with certain hardware modifications. PDP-11/40 is a predecessor of PDP-11/45 and obviously presents more problems. Main differences in Virtualization of the two models of PDP-11 have been discussed in Appendix A.

As mentioned before, the PDP-11/40 machine available to us did not have any software support. Therefore, as a first step of the project, a small UNIX [RiT74] like operating system called MUNI (mnemonic for Mini UNIX) was developed.

A brief description of MUNI is given in Appendix B.

The structure of this report is as follows. Chapter II discusses PDP-11 I/O subsystem architecture and describes main elements of VMM-11 I/O module. Chapter III describes individual devices available on VMM-11. The problems encountered in the implementation (some solved and some still unsolved) have been discussed in Chapter IV. Chapter V summarizes and concludes the report. There are 5 Appendices, and as mentioned before, Appendix A describes difference between PDP-11/40 and PDP-11/45 virtualization, and Appendix B is devoted to the mini monitor MUNI used for VMM development. The VMM kernel, which currently does VM scheduling, has been described in Appendix C. Appendix D is devoted to programming of X-BUS. Appendix E is entitled, "On Control Theoretic Approach to Memory Management."



## CHAPTER II

### TECHNIQUES OF I/O HANDLING IN VMM-11

#### 2.1 OVERVIEW :

There are 4 main modules of a Virtual Machine Monitor as shown in Figure 2.1, viz. :

1. VMM kernel : For inter-process communication and security, process scheduling, and interrupt/trap handling.
2. I/O module : To provide I/O to real and virtual devices.
3. Instruction simulation module : To trap and simulate sensitive instructions.
4. Panel simulation module : To provide console panels for Virtual Machines.

Of these, we have already completed I/O module and a part of VMM kernel. Absence of Memory Management Unit has prevented us from proceeding further on other modules. The VMM kernel has been described in Appendix C. In this chapter, we discuss those details of I/O module which do not depend upon individual I/O devices, such as trapping of the I/O request, communication of request to PDP-10 and various data structures used in the process.

#### 2.2 STRUCTURE OF I/O MODULE :

As shown in Figure 2.2, I/O module consists of the following

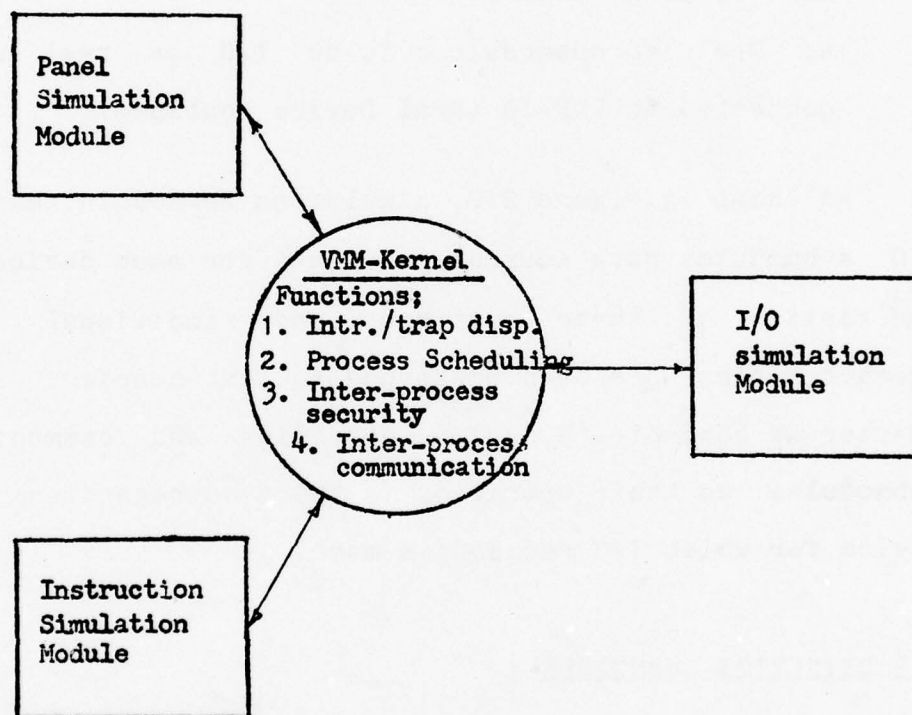


Figure 2.1 : Principal Modules of a Virtual Machine Monitor.

submodules :

1. Detection submodule : For trapping of I/O request.
2. Simulation submodule : To simulate I/O to individual devices (Virtual Device routines).
3. Communication submodule : To transfer requests to PDP-10, if necessary.
4. Real I/O submodule : To do I/O on real devices connected to PDP-10 (Real Device routines).

As shown in Figure 2.2, simulation submodule, and Real I/O submodules have separate routines for each device. The description of these routines and individual device characteristics has been postponed to next chapter. In this chapter we concentrate on I/O detection and communication submodules as their operation is the same regardless of the device for which I/O request is made.

### 2.3 DETECTION SUBMODULE :

PDP-11 has a UNIBUS type of I/O subsystem architecture. In this architecture, channel programs for each device are hardwired in the device controller and the parameters for these programs are supplied by writing them in fixed I/O locations. Thus each I/O device has a set of I/O registers. All such registers are located in upper 4K of memory.

Thus if the upper 4K of memory is rendered non-existent using the Memory Management unit, any attempt to do I/O can

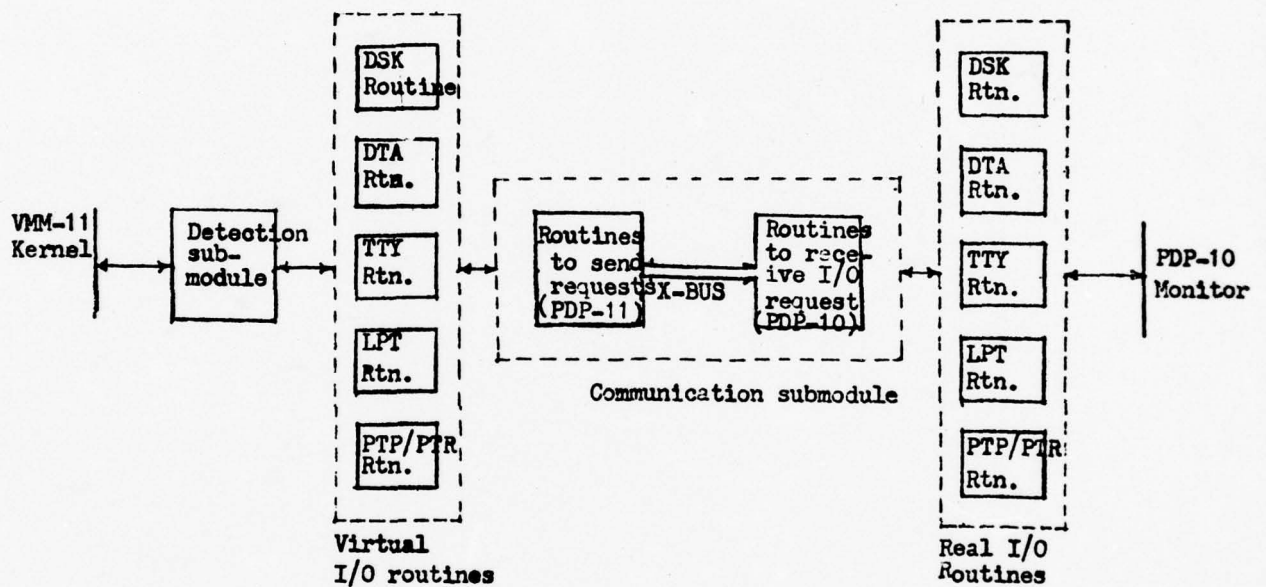


Figure 2.2 : Structure of VMM-11 I/O module.



be trapped whenever the user tries to access these I/O registers. The bus error handling routine can then supply the following information to VMM I/O routines :

1. Address of location being accessed.
2. Operation being attempted (reading or writing).
3. If operation is writing, the data being written.

The VMM I/O routines then call appropriate virtual device routine using a table of I/O location.

2.3.1 Table of I/O Locations : PDP-11 I/O registers are peculiar in the sense that they have read only bits, write only bits, and I/O responsive bits distributed in a non standard fashion. (A bit is called I/O responsive if its reading/writing results in either an I/O operation or changing of other bits).

VMM-11, therefore, maintains a table of PDP-11 I/O registers and their characteristics as shown in Figure 2.3 . Each entry in the table is 8 words long and contains the following information :

1. Location Address
2. Data in that location
3. Readable bit mask (or Read-mask)
4. Writable bit mask (or Write-mask)
5. Responsive bit mask
6. Address of device handling routine

	Location Address	Content	Read mask	Write mask	Sens. mask	Routine address	Initial Un- Content used.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
Punch Status	177554	200	100300	100	0	0	200 0
Punch Data	177556	0	0	377	-1	PTPRTN	0 0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.

Figure 2.3 : Location table for PDP-11 I/O registers.

7. Initial Content (Content after 'INIT' operation)
8. Reserved for future use

As an example, actual entries corresponding to High Speed Paper Tape Punch are also shown in Figure 2.3 . A one in the mask indicates that the corresponding bit is readable/writable/responsive. Normally setting a responsive bit to 1 initiates I/O operation. However, there are cases when even reading or clearing of a particular location is I/O responsive. Such locations are given a responsiveness mask of -1.

On a read request the content of the register is ANDed with the read mask and the result is passed on to VM requesting it. Similarly, on a write request, the write mask specifies writable bits. In both cases a check is made to ensure that no I/O responsive bit has been affected and if it is, an appropriate routine is called. This routine will do the requested I/O, or if necessary pass the request to PDP-10, and set the appropriate condition bits in the device status register. Also it will indicate to VMM-11 if an interrupt is to be simulated.

#### 2.4 COMMUNICATION SUBMODULE :

A request is communicated to PDP-10 by putting a Request Control Block (RCB) in a queue of requests to be read by PDP-10. The RCB can be put in one of 8 priority queues as shown in Figure 2.4. The priority assigned to a request is

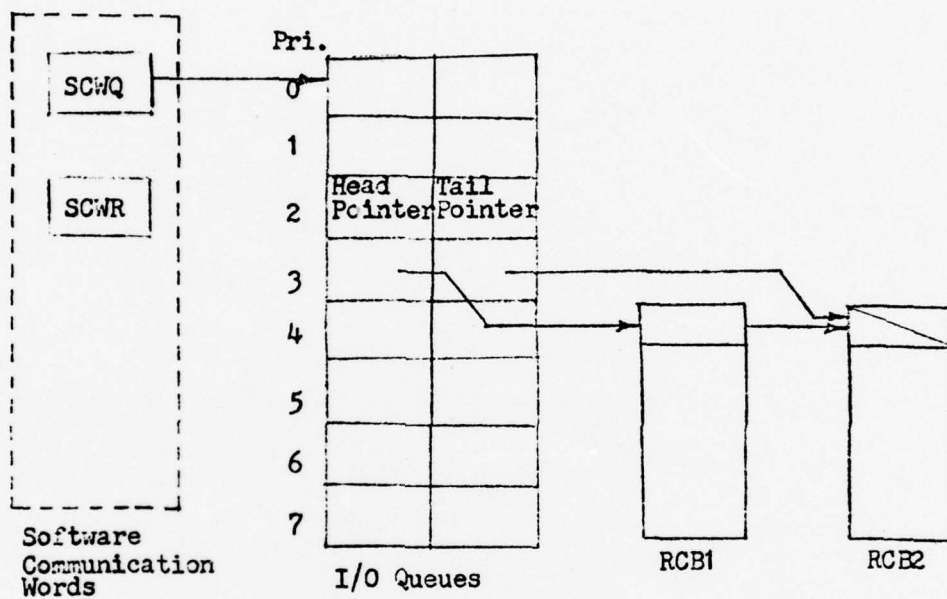


Figure 2.4 : I/O Queues for PDP-10 devices



decided by speed of the device. Requests are served by PDP-10 in the order of priority i.e., highest priority first. Request at the same priority level are served using a first come first served discipline. Recall that our aim in developing the VMM is to be able to experiment with resource management techniques. Hence, the service disciplines such as the one described here and in other parts of this report are temporary. These will be changed from time to time for experimentation.

2.4.1 Request Control Block : Each I/O request has a fixed format Request Control Block (RCB) as shown in Figure 2.5.

Main elements of RCB are :

1. Link word : Points to next RCB in the queue.
2. Priority : Priority of this I/O request.
3. Done flag : Indicates that this request has been serviced by PDP-10. (RCB remains in the queue until removed by PDP-11)
4. Semaphore : Indicates semaphore number to which the process waiting for this I/O is queued. A V operation will be performed on this semaphore after the request has been serviced.
5. Device type : Each device has been assigned a number. For example, device 3 is Paper Tape Reader.
6. Pointer to DDM : This points to a Device Dependent Message (DDM) which varies from device to device. For example, in case of Teletype DDM consists only of

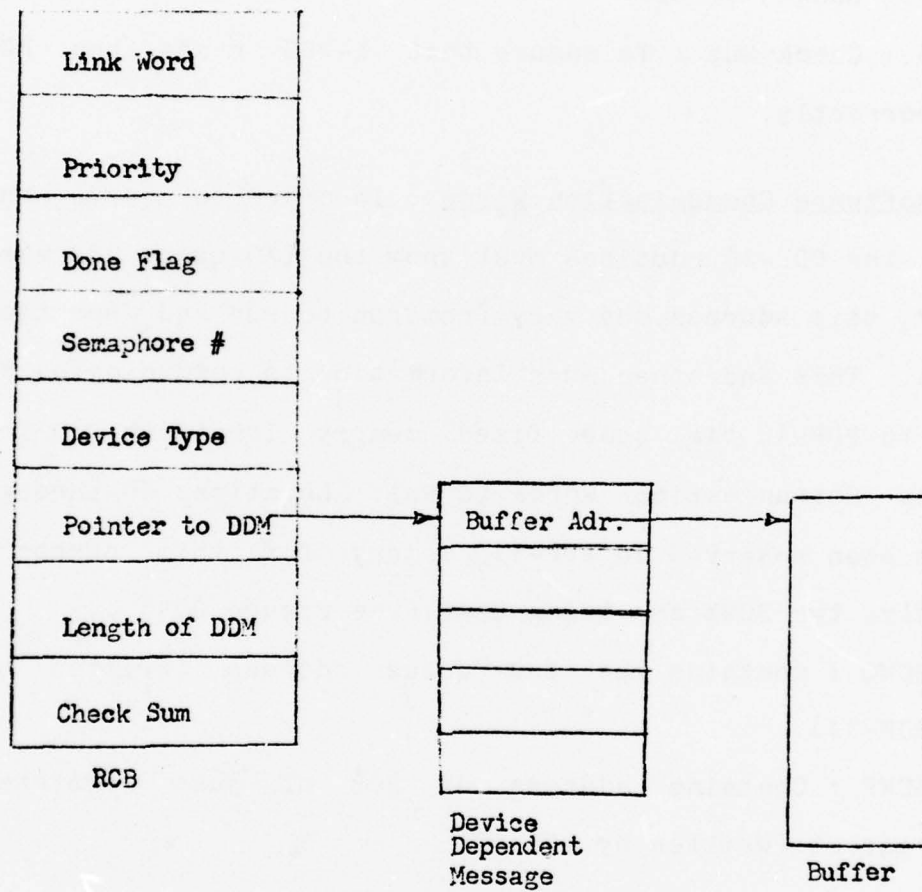


Figure 2.5: Request Control Block

buffer address and buffer size while in the case of Disk it contains current values of 6 Disk I/O registers. DDMs for various virtual devices have been described in next chapter.

7. Length of DDM

8. Check Sum : To ensure that X-BUS reads the RCB correctly.

2.4.2 Software Communication Words : In order to serve the PDP-11, the PDP-10 routines must know the I/O queue address. However, this address may vary from run to run and from time to time. This and other such information is communicated by PDP-11 to PDP-10 via some fixed memory locations called Software Communication Words (SCWs). Locations 40 through 56 have been reserved in PDP-11 memory for this purpose. Currently, two SCWs are being used (see Figure 2.4) .

SCWQ : Contains the I/O queue address (written by PDP-11)

SCWR : Contains address of RCB of just completed request (written by PDP-10).

After the PDP-10 has serviced a request, it sets the done flag in the RCB, writes the address of the RCB in SCWR and interrupts PDP-11 at location 144 (PDP-11 can be interrupted by PDP-10 at 8 different interrupt locations, namely, 140, 144, 150, 154, 160, 164, 170, and 174). The interrupt servicing routine then reads SCWR, removes the RCB

from I/O queue, and performs a V operation on appropriate semaphore to wake the process that submitted this request.



CHAPTER III  
DESCRIPTION OF VIRTUAL I/O DEVICES IN VMM-11

In previous chapter, we have discussed those details of I/O simulation module which are common to all virtual devices, such as, techniques of detecting I/O requests, I/O service discipline, and structure of request control blocks, etc. In this chapter, we examine the virtual devices one by one and explain the implementation of their functions.

VMM-11 provides virtual I/O facilities for following devices:

1. One DECpack Disk Cartridge RK-11.
2. Eight TC-11 DECTape units.
3. One LP-11 Line Printer unit.
4. One High Speed Paper Tape Reader/Punch PC-11.
5. One LT-33 Teletype (in addition to the Teletype connected to PDP-11).

Model numbers of virtual devices have also been given above because different model numbers of same device use different I/O register locations. For example, RPJ-11 disk has a complete separate set of I/O registers than RK-11. As mentioned before, The real devices on the PDP-10 can be different from corresponding virtual devices. Some discussion relating to functions available on these devices in VMM-11 follows.

### 3.1 RK-11 DECPACK DISK CARTRIDGE :

There are several different types of disks available to a real PDP-11 user. Of these, RK-11 disk pack has been chosen for implementation in VMM-11 mainly for reasons of it's simplicity and size. It can be used by manipulating bits of following 6 I/O register locations:

<u>Location</u>	<u>Mnemonic</u>	<u>Function</u>
177400	RKDS	Drive status register
177402	RKER	Error register
177404	RKCS	Control and status register
177406	RKWC	Word count register
177410	RKBA	Current bus address register
177412	RKDA	Disk address register

Functions of the various bits (some of which are read only, and some write only) in these registers are described in PDP-11 Peripherals Handbook. In essence these provide following functions:

1. Control Reset
2. Write
3. Read
4. Write Check
5. Read Check
6. Seek
7. Drive Reset
8. Write Lock

Of these functions, write and read requests are transmitted to PDP-10 portion of VMM-11 which does the real I/O to a file named "PDP11.DSK". Control reset, drive reset, write lock, and seek are software simulated in PDP-11 itself. For example, on a 'seek', the cylinder number sought is compared with the currently defined size of the disc. A seek error bit is set if the cylinder is beyond the capacity, otherwise, a successful return is given, a record is kept of current cylinder sought, and is used for next read/write operation. For the remaining two functions namely, read check, and write check we have not been able to get full definitions. Thus currently, VMM-11 gives immediate successful return on these functions. The Device Dependent Message (DDM) for RK-11 is 7 words long and contains current values of 6 Disk I/O registers, and a check sum word.

### 3.2 TC-11 DECTAPES :

VMM-11 provides virtual I/O facility to all 8 DECTape units available on PDP-10. An "ASSIGN" command at monitor level must be used before starting VMM-11 to define available units. For example, a command:

ASSIGN DTA6:DT2

declares that physical DECTape unit #5 is to be used as 2nd unit for PDP-11 user running on VMM-11.

DECTapes can be used via following 5 I/O registers:

<u>Location</u>	<u>Mnemonic</u>	<u>Function</u>
177340	TCST	Control and status register
177342	TCCM	Command register
177344	TCWC	Word count register
177346	TCBA	Bus address register
177350	TCDT	Data register

There are 8 commands available to TC-11 users:

1. SAT            Stop all tape motion
2. RNUM          Read block number
3. RDATA        Read data
4. RALL          Read all
5. SST           Stop selected transport only
6. WRTM         Write track and mark
7. WDATA        ~~Write data~~
8. WALL          Write all

Of these SAT, RNUM, and SST are software simulated inside VMM-11. This saves a lot of overhead. For example, to seek a desired block number, a PDP-11 user has to specify direction of motion and issue a series of 'RNUM' commands to read successive block numbers. A read/write command is issued when the desired block is reached. In VMM-11, this same effect is achieved by maintaining a table of current block number and direction of motion for each unit. On RNUM the block # is incremented/decremented by one depending on direction of motion, and passed on to user. Finally when the user issues a RDATA/WDATA command, the PDP-10 portion of



VMM-11 is informed to seek the current block number and do the I/O. Thus, numerous tape start/stops at individual RNUMs are avoided. Similarly, on a SAT (stop all tape motion) or SST (stop selected transport only), directions of motion of all or selected tape unit are set to zero.

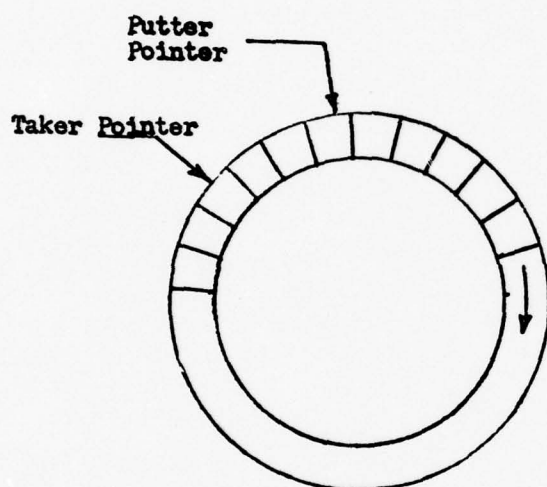
Request for reading and writing data (RDATA, and WDATA) are passed on to PDP-10 portions of VMM-11 and actual I/O is done to corresponding tape unit. Function WRTM (write track mark) is not available to VMM-11 users because there is no provision for this facility on PDP-10 monitor. Remaining, two functions RALL, and WALL are unimplemented due to our inability to get full definitions of these functions. The DDM for DECTapes is similar to that for Disk. It is 6 words long and contains current value of 5 I/O registers and a check sum word.

### 3.3 LP-11 LINE PRINTER :

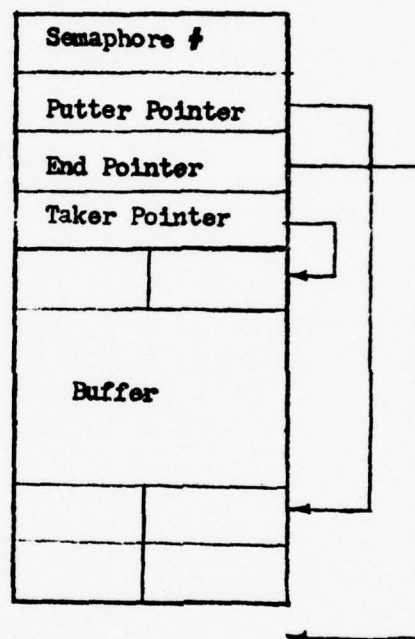
LP-11 is a character oriented device. That is, the user writes on to it character by character, by writing the characters on buffer register LPDB. The two I/O registers associated with LP-11 are:

<u>Address</u>	<u>Mnemonic</u>	<u>Function</u>
177514	LPCS	Control and status register
177516	LPDB	Data buffer

VMM-11 maintains an internal circular buffer for LPT (see Figure 3.1). The putter pointer is manipulated by PDP-11



a : Logical View



b ; Physical Organization

Figure 3.1 : Circular buffer for byte oriented devices.

and taker pointer by PDP-10. The characters written on LPDB are saved in this buffer, the putter pointer is advanced, and a successful return is given. When the buffer becomes full (i.e. when putter is one word behind the taker), a request is sent to PDP-10 portion of VMM-11, which then empties the buffer, resets the taker pointer, and writes data on to the spooled LPT. DDM for LP-11 consists of only 2 words- buffer address, and buffer size. Range checking is used to ensure correctness.

#### 3.4 PC-11 PAPER TAPE READER/PUNCH :

Handling of PC-11 is very similar to that for Line Printer. VMM-11 maintains two circular buffers, one for Reader and another for Punch. Requests for filling/emptying the buffer are transmitted to PDP-10 whenever required. DDMs for Reader, and Punch are separate and are similar to that of LP-11.

#### 3.5 TELETYPE :

VMM-11 handles TELETYPE in the same manner as Paper Tape Reader/Punch. Any TELETYPE connected to PDP-10 can be used for VMM-11 TTY I/O by assigning it a logical name "TTK".

CHAPTER IV  
PROBLEMS ENCOUNTERED DURING DEVELOPMENT

This chapter is devoted to discussion of problems that were encountered during development of Virtual I/O module of VMM-11. Some of these problems have been solved and some are yet to be solved. The problems center around the X-BUS and Memory Management unit as discussed below.

4.1 MULFUNCTIONING OF X-BUS :

Some problems were encountered in communication between PDP-10 and PDP-11 portions of VMM-11. Initially, we thought that the problems were due to simultaneous access of the I/O queue. The queue of I/O requests is a resource that is used by both PDP-10 and PDP-11 portions of VMM-11. In order that the PDP-10 may not read the queue at a time when PDP-11 is modifying it, it was suggested that semaphore technique be used for mutual exclusion. Therefore, routines were written which allowed PDP-10 and PDP-11 to get exclusive control of I/O queue. However, this did not solve the problem. In fact, the overhead introduced by these routines made the problem worse.

Finally, after a lot of experimentation, it was discovered that root of the problem was the X-BUS. It reads/writes memory correctly only if PDP-11 processor is in wait mode. But, if processor is running (and hence trying to access memory) X-BUS misses words. The missing frequency



can be as high as 5000 ERRORS PER MILLION. This is quite serious, considering that we make 9000 reads per minute on the X-BUS and one unnoticed error can be enough to bring our system down. Anyway, all communication routines on PDP-11 and PDP-10 have now been modified to check the correctness of each message received before using it. Checking techniques include check summing and range checking. If any error is detected, the message is reread and the cycle repeated. VMM-11 also keeps statistics on the number of errors encountered. Missing frequency quoted above was obtained from VMM-11. The X-BUS hardware is being re-examined to correct malfunctioning.

#### 4.2 NEED FOR MEMORY MANAGEMENT UNIT :

The absence of Memory Management unit (MMU) has been the main obstacle in our progress. The PDP-11 without MMU is not virtualizable. However, in our development of the Virtual I/O module we have gone as far as we could without having an MMU. Some portions of the I/O module which depend on MMU have been temporarily simulated to demonstrate the feasibility of our approach. Without an MMU, it is almost impossible to proceed on inter-process security, and instruction simulation modules. In case of I/O virtualization, bus error handling and interrupt simulation routines can not be written without MMU as explained below.

4.2.1 Bus Errors Handling: A PDP-11 instruction can be 1, 2, or 3 words long. The program counter register PC is incremented successively as 1st, 2nd, and 3rd words of the instruction are fetched. Thus when a bus error occurs, it is not possible to exactly determine where the instruction began and hence, what address it was trying to access. The only remedy is to have a Memory Management unit. The unit has a special register to hold beginning address of each instruction.

The Memory Management unit is also needed to make all I/O registers non-existent, so that any attempt by user to access these location will signal VMM-11. Currently, these locations are directly accessible to user and VMM-11 knows of I/O requests only when the user explicitly tells it.

4.2.2 Interrupt Simulation : Currently, VMM-11 is unable to allow the user to do I/O in asynchronous mode using interrupts. The user is blocked from progressing as soon as it makes an I/O request, and is allowed to progress only after the I/O is complete. This is because there is no way for VMM-11 to keep track of the user PSW and to interrupt it when priority is below hardware defined priority of the I/O device. The only solution is to use a Memory Management unit to keep track of user PSW. Thus interrupt simulation has been postponed pending arrival of Memory Management unit.

. CHAPTER V  
SUMMARY

A Virtual Machine Monitor for PDP-11/40 mini-computer is being developed to demonstrate usefulness of Virtual Machine techniques. One aim of this VMM is to be able to measure performance of new resource management techniques being developed here for operating system optimization. Virtual I/O module, and a portion of VMM kernel have already been developed. In addition, a small monitor MUNI has been written to aid in VMM software development.

Even though our PDP-11/40 computer does not have I/O devices connected to it, VMM-11 provides Virtual I/O to an RK-11 DECpack Disk Cartirdge, a TC-11 tape controller with 8 DECTape units, an LP-11 Line Printer unit, a PC-11 High Speed Paper Tape Reader/Punch, and a remote LT-33 TELETYPE. Also VMM-11 provides multiple- VM structures so that these virtual devices can be divided among various VMs in any desired manner. Thus, we have successfully demonstrated feasibility of providing Virtual I/O support using a Virtual Machine and a hardware link to a resourceful computer system.

Some problem were initially encountered due to mulfunctioning of the X-BUS. These have now been overcome by using software checks. Work on other modules of VMM-11

is still continuing. However, absence of Memory Management Unit is becoming an obstacle to further progress of our research.



## VI. REFERENCES

- [Arn76] C. R. Arnold, "Optimization of Computer Operating Systems", Ph. D, Thesis, DEAP, Harvard University, Cambridge, MA 1976 (to be submitted).
- [BuG73A] J. P. Buzen, and U. O. Gagliardi, "The evolution of virtual machine architecture", AFIPS Conference Proceedings, NCC 1973.
- [BuG73B] J. P. Buzen, and U. O. Gagliardi, "Introduction to Virtual Machines", Honeywell Computer Journal, Vol. 7, No. 4, 1973.
- [BuG74] J. P. Buzen, and R. P. Goldberg, "Virtual Machine Techniques for Introducing Peripherals into Computer Systems", Computer Peripherals - Benefactor or Bottleneck? Digest of Papers COMPCON 74, San Fransisco, February 1974, pp. 157-160.
- [Fra73] C. Frasson, "Simulation of Input-Output Units", Proceedings ACM SIGARCH - SIGOPS Workshop on Virtual Computer Systems, Cambridge, MA 1973.
- [GaG72] U. O. Gagliardi, and R. P. Goldberg, "Virtualizable Architectures", Proceeding of 1972 ACM AICA International Comp. Symp., Venice, Italy, April 1972, pp. 527-538.
- [Gol72] R. P. Goldberg, "Architectural Principles for Virtual Computer Systems", Ph.D. Thesis, DEAP, Harvard University, Campridge, MA 1972.
- [PoK76] G. J. Popek, and C. J. Kline, "The PDP-11 Virtual

Machine Architecture : A Case Study", To be published.

[RiT74] D. M. Ritchie, and K. Thompson, "The UNIX Time-sharing System", CACM Vol. 17, No. 7, July 1974, pp. 365-375

APPENDIX A  
PDP-11/40 VIRTUALIZATION

The condition for virtualization of an architecture have been discussed in detail by Goldberg in his Ph.D thesis [Gol72]. Using these conditions, Popek & Kline [PoK76] have pointed out that the main obstacles to virtualization of PDP-11/45 are following:

1. There are 10 sensitive instructions which do not trap properly. These are  
RTI, RTT, RESET, WAIT, HALT, MTPI, MFPI, MTPD, MFPD, AND SPL.  
For definition of sensitive instruction, reader is referred to [Gol72].
2. There is excessive overhead in I/O simulation due to UNIBUS architecture wherein I/O registers are treated as memory locations.
3. Some bits of I/O registers are read-only, some are write-only, and these appear randomly in the word, thus creating severe protection simulation problems.

PDP-11/40 is a predecessor of PDP-11/45. Its virtualization, therefore, presents not only the above difficulties but many more. The main architectural differences between the two models and their influence, if any on virtualization of PDP-11/40 are discussed below.

1. In the PDP-11/40 there are only two modes (kernel & user) as compared to three (kernel, supervisor, and user) in PDP-11/45. Thus Popek's suggestion of implementing security nucleus in kernel mode and the remainder of VMM in supervisor mode is not feasible. Whole VMM has to be implemented in kernel mode. This will influence the design of interprocess security feature of VMM.
2. The Memory Management unit of the PDP-11/40 does not distinguish between instruction space and data space. Whole space is treated as instruction space. Thus there are only two M-series instructions, namely MTPI (Move To Previous Instruction space) and MFPI (Move From Previous Instruction space), as opposed to four in PDP-11/45. This again will influence the design of interprocess communication and security features of VMM-11.
3. The PDP-11/40 processor has four priority level as opposed to eight in PDP-11/45, and there is no SPL (Set Priority Level) instruction. In kernel mode, if desired, the PSW can be treated like any other memory location. In user mode, programs are inhibited from changing priority level and mode bits of PSW, and the only way of changing these is by traps and interrupts. However, we do not yet know whether and where the processor traps on such an attempt.
4. The basic processor (KD-11A) used in PDP-11/40 does not have MUL, DIV, ASH (Arithmetic Shift), and ASHC (Arithmetic



Shift Combining two registers). However, these instructions are not sensitive and should not make any difference in virtualization.

5. The PDP-11/45 has two sets of 8 general purpose registers. Thus one set can be used by user programs and another by supervisory programs. The PDP-11/40 has only one set of 8 registers, thus requiring user registers to be saved before they can be used by supervisory programs.

6. In the PDP-11/45 during each instruction execution, Status Register #1 (SR1) keeps track of auto-increments or auto-decrements of general registers. This helps in recovering from page faults. SR1 has not been implemented in PDP-11/40 Memory Management unit. Hence, many times it is not possible to recover from page faults.

## APPENDIX B

### MUNI : THE MINI UNIX LIKE OPERATING SYSTEM

In developing the Virtual Machine Monitor, we had to start from scratch. The PDP-11/40 computer available to us did not have any software support. Therefore, a small operating system called MUNI (mnemonic for Mini UNIX) was quickly developed. Similarity to UNIX [Ri74] was especially chosen so that program developed on Harvard undergraduate computing facility (a PDP-11/45 with UNIX) could be run on our system without change. All VMM programs were developed and debugged initially on UNIX and final modification and debugging was done on MUNI. Currently, MUNI provides complete TTY I/O, 6 most commonly used system calls, and an on line debugging utility.

#### B.1 SYSTEM CALLS :

The system calls provided by MUNI are listed below

- \$READ : To read from TTY
- \$WRITE : To write on TTY
- \$EXIT : To exit from program to monitor
- \$STTY : To set TTY modes
- \$GTTY : To get current TTY mode
- \$SIG : To specify user's signal handling routines

The format of these system calls is exactly the same as in UNIX. For example, in order to write 17 characters on

TTY starting from address "LOC" the user's assembler program should call

\$WRITE

LOC

17.

Similarly, with the help of \$SIG system call, the user can specify his own routines to handle control-C interrupt, control-B interrupt, EMT trap, T-bit trap, IOT trap, and bus error etc.

#### B.2 ON LINE DEBUGGING :

Even if a program has been thoroughly debugged on UNIX, very often one finds that it does not work on PDP-11/40. The reasons may be an error in reading PDP-11 tapes on PDP-10, error in cross assembling on PDP-10, or error in loading the program via X-BUS. The ODT package helps the user debug his program on PDP-11/40 directly.\* Again ODT is standard UNIX ODT. The details of ODT commands can be found in DEC's PDP-11 Paper Tape System Manual.

---

\* It may be of some interest to note here that a cross-ODT has also been developed and included in PDP-10 portion of VMM-11 code. It runs on PDP-10 and allows user to peek into PDP-11. Its commands are very similar to those of ODT. It has been very useful in debugging X-BUS and also VMM-11.

### B.3 TTY I/O :

MUNI provides complete TTY I/O facility. Using \$STTY system call user program can set TTY to raw/line, echo/noecho and other modes. In line mode, TTY input is passed on the user program only when a complete line has been typed in. Thus, user can edit the line. In raw mode, on the other hand, every character is passed on to user program as soon as it is typed in. Also, like most other PDP operating systems MUNI understands following special characters on typein :

Control-C = Immediate exit to monitor

Rubout = Rubout last typed in character

Control-U = Delete last typed in line

Control-O = Suppress further typeout from program.

In addition, if TTY mode is appropriately set, line-feed and carriage return can be echoed as newline (i.e., <CR> <LF>) and all control characters can be echoed in Arrow-Character format. A newline is automatically generated when TTY head reaches to the end of TTY line. A bell rings when TTY input buffer is full and no further input can be accepted.

### B.4 MUNI IMPLEMENTATION :

The code for MUNI is highly modular. New commands and system calls can be added by simply entering them in command tables. The code is full of comments. For ease of debugging, subroutines have been used as often as possible,



so that one can bypass those routines which are known to be working. Inclusion of new utilities (like assembler, text editor etc.) is particularly easy. ODT, for example, makes use of only 6 system calls mentioned above and was easily included into MUNI.

APPENDIX C  
CURRENT STATUS OF VMM-11 KERNEL

The Kernel is the nucleus of the Virtual Machine Monitor. Its primary functions are:

1. Interrupt/Trap dispatching to proper modules
2. Process scheduling.
3. Inter-process security.
4. Inter-process communication.

VMM-11 allows multiple virtual machines and therefore inter-process functions mentioned above are very important. The kernel is still under development. Currently, it provides interrupt/trap dispatching and process scheduling. Absence of Memory Management Unit has been an obstacle to our progress on inter-process security and inter-process communication.

Before we proceed to describe process scheduling and synchronization in VMM-11, it should be reiterated that the main aim of VMM-11 is to enable us to experiment with different resource management and scheduling techniques, and hence, the queue disciplines and data structures used herein will keep changing from time to time. Initially, we have chosen a very simple first come first served discipline and a minimal process control block. However, the code has been kept very flexible so that extentions/modifications of these

would be easy.

### C.1 PROCESS SCHEDULING :

VMM-11 allows multiple virtual machines. Each VM is treated as a separate process. Dijkstra's semaphore technique with well known P and V operations is used for process synchronization. All processes that are ready to run and are not waiting for any resource are put in one of several run queues. Each process has an assigned software priority. For each priority there is a run queue. A process gets a chance to run only if no processes with higher priorities are waiting. A running process is blocked and put back in run queue if a higher priority process becomes ready to run.

### C.2 RUN QUEUES :

In VMM-11, run queues are maintained as a queue table as shown in Figure C.1. For each priority there are two entries in the table, namely head pointer and tail pointer. The head pointer points to Process Control Block (PCB) of first process and tail pointer points to PCB of last process in the queue. All PCBs in a queue are singly linked and are serviced by first come first served discipline.

### C.3 PROCESS CONTROL BLOCKS :

As shown in Figure C.2 the essential elements of the PCB are:

1. Link word : Points to next PCB in the queue.

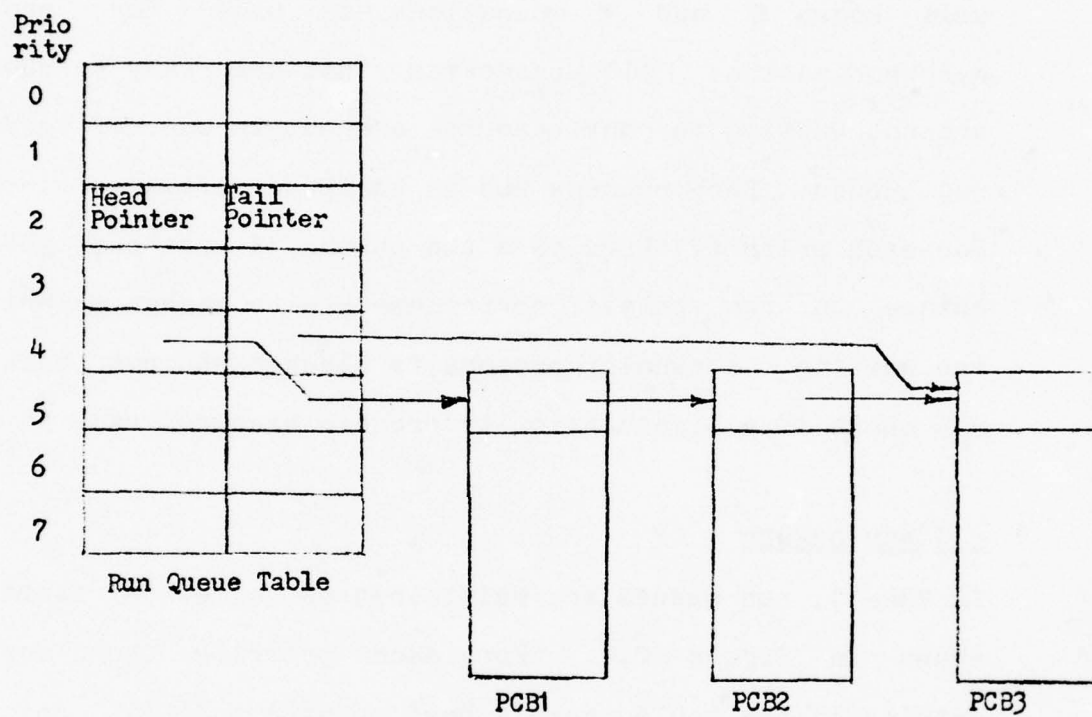


Figure C.1 : Run queues



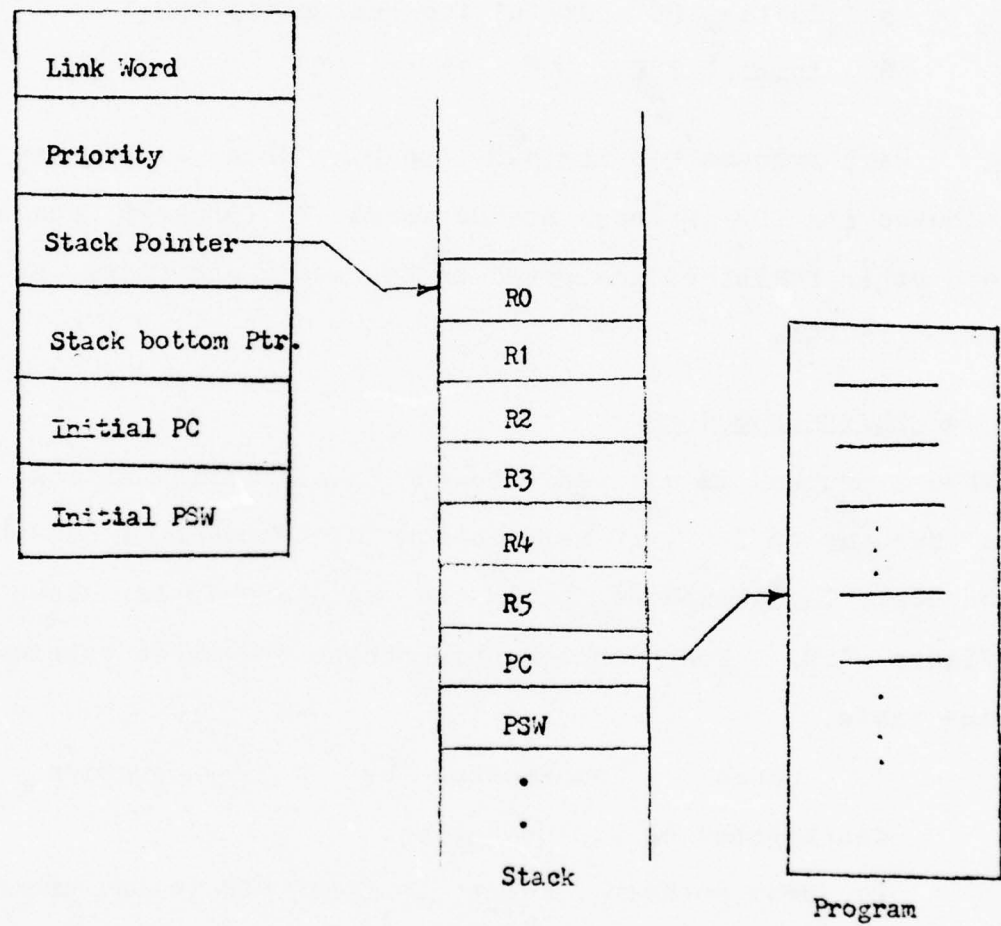


Figure C.2 : Process Control Block

2. Priority : Software priority of the process.
3. SP : Stack pointer.
4. Stack Bottom Pointer : Points to the highest allowable value of SP (for security).
5. Initial PC : Useful for restarting VMM-11
6. Initial PSW : " " " "

Each process has its own stack. When a process is blocked its PSW (Process Status Word), PC (Program Counter), and other registers are saved on its stack and final SP is saved in PCB.

#### C.4 SEMAPHORE QUEUES :

When a process is not ready to run (i.e. it is waiting for a resource or I/O), it waits at one of a number of semaphore queues. The semaphore table is maintained as shown in Figure C.3. For each semaphore there are three entries in the table.

1. Count : Incremented by a V-operation and decremented by a P-operation.
2. Head pointer : Points to first PCB in the queue.
3. Tail pointer : Points to last PCB in the queue.

Again a first-come-first-served discipline is used to free processes from semaphore queues.

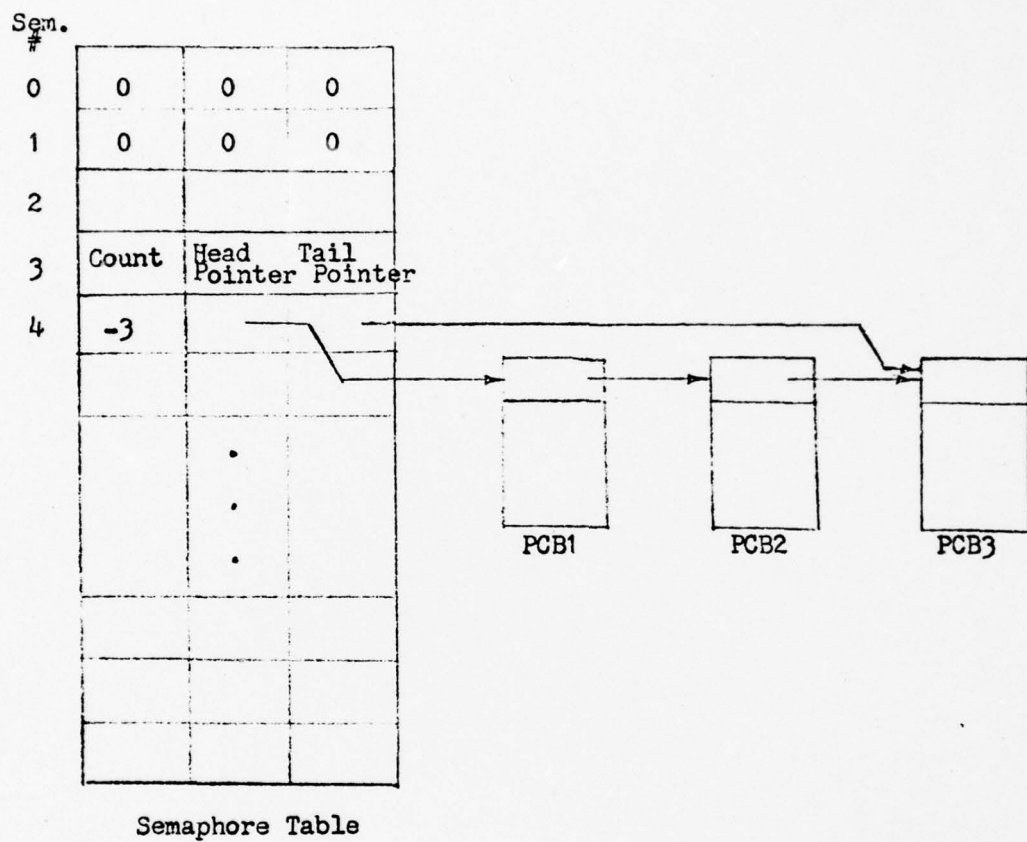


Figure C.3 : Semaphore queues

## APPENDIX D

### PROGRAMMING THE X-BUS

X-BUS is the hardware link connecting PDP-11 and PDP-10. It provides a master-slave relationship between the PDP-10 and the PDP-11. To be more precise, the PDP-10 can use the X-BUS to read/write the PDP-11 memory, to start/stop/interrupt the PDP-11, or to sense status of PDP-11. The PDP-11 can not use it the same way to control the PDP-10. However, if PDP-10 has set appropriate bits, the PDP-11 can interrupt the PDP-10. The usual way for the PDP-11 to communicate with PDP-10 is to write it in a buffer and wait till PDP-10 reads it. For the PDP-10 X-BUS is just another I/O device. DECsystem-10 monitor provides I/O facilities to the X-BUS in a manner similar to other devices, i.e., using IN, OUT UUOs and buffer rings. However, as will be explained later, the monitor should be explicitly requested to set up only single buffer rings (a two buffer ring is default for all devices including the X-BUS), using INBUF and OUTBUF UUOs.

#### D.1 X-BUS FUNCTIONS :

X-BUS interface is designed for following 7 functions :

1. LOAD ADDRESS in the memory address register of PDP-11
2. WRITE DATA in the PDP-11 memory.



3. READ DATA from PDP-11 memory.
4. SELECT DEVICE (#1 = PDP-11/10, 2=PDP-11/40)
5. CONTROL PDP-11
6. SENSE STATUS of PDP-11
7. SENSE INTERRUPT from PDP-11.

There are two PDP-11's (a PDP-11/40, and a PDP-11/10) connected by the same BUS to PDP-10. The function SELECT DEVICE enables the X-BUS to choose one of the two PDP-11's for subsequent commands.

Meaning of WRITE DATA, and READ DATA is quite obvious. However, before reading/writing, desired address is specified by LOAD ADDRESS function.

The function CONTROL DEVICE enables PDP-10 to control PDP-11's in the following manner :

1. Console switch functions like ENABLE/HALT, LOAD ADDRESS, and START can be done directly from PDP-10.
2. PDP-11's can be interrupted at one of eight possible interrupt locations.
3. PDP-11 can be preempted to enable the X-BUS to become master of UNIBUS even if PDP-11 is halted.
4. PDP-10 can be interrupted whenever PDP-11 comes to a halt.

The function SENSE STATUS allows the PDP-10 to find out whether the PDP-11 is running, whether its HALT switch is

down, and whether its power is up. Also it allows PDP-10 to determine what was the cause of interrupt to it from PDP-11.

The SENSE INTERRUPT function allows PDP-11 to interrupt PDP-10. However, software necessary to handle interrupts from PDP-11 does not yet exist in DECsystem-10 monitor. Therefore, currently all interrupts to PDP-10 are ignored.

#### D.2 CONTROL AND DATA WORDS :

Every 36 bit word sent to BUS from PDP-10 is interpreted as made up of two 18 bits half words. The left half is taken as control word to the X-BUS interface, and other half is data to be passed. Figure D.1 and Table D-1 explain the assignment of bits in control half.

Bits in data half of words are interpreted differently in different modes as shown below:

<u>Mode</u>	<u>Mnemonic</u>	<u>Meaning of data half</u>
1	SENSE INTERRUPT	Ignored
2	SENSE STATUS	Ignored
3	READ DATA	Address to be read
4	WRITE DATA	Data to be written
5	SELECT DEVICE	Device number
6	LOAD ADDRESS	Address to be loaded
7	CONTROL DEVICE	Control function.

The function CONTROL DEVICE loads the associated data half into a PDP-11 control register, whose bits assignment is shown in Figure D.2 and explained in Table D-2.

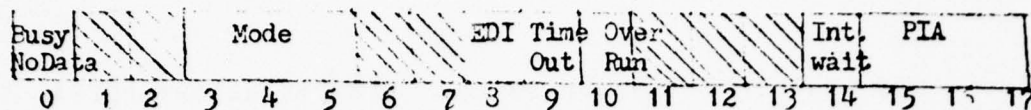


Figure D.1 : Bit assignment in Control Half

TABLE D-1 : Bit assignment in Control Half.

Bits	Mnemonic	Description
0	Busy/No data	On input (CONI) : Bus is busy On output(CONO) : No data in right half of this word.
3-5	Mode	Function to be performed : 0 = None 1 = Sense Interrupt 2 = Sense Status 3 = Read Data 4 = Write Data 5 = Select Device 6 = Load Address 7 = Control Device
8	EDI	Enable Done Interrupt : Interrupt PDP-10 on function completion.
9	Time out	On input (CONI) : Time out occurred during last operation. On output(CONO) : Clear time out bit.
10	Over run	On input (CONI) : An over run condition was sensed during last operation. On output(CONO) : Clear Over run bit.
14	Interrupt Wait	Bus is waiting to interrupt PDP-11.
15-17	P.I.A.	Priority Interrupt Assignment : must be zero.

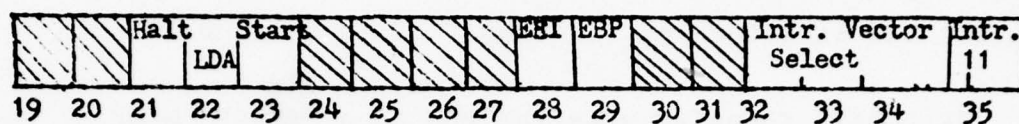


Figure D.2 : Control function bit assignment.

TABLE D-2 : Explanation of Control Function bits.

Bits	Mnemonic	Description
22	HALT	Stop PDP-11
23	LDA	Load Address : Load console switch register into memory address register.
24	Start	Start PDP-11 program from current address.
28	EHI	Enable Halt Interrupt : Interrupt PDP-10 if PDP-11 comes to a halt.
29	EBP	Enable Bus Preempt : Preempt PDP-11 and enable X-BUS to become master of UNIBUS even if PDP-11 is halted.
32-34	Int. Vec. Sel.	Interrupt Vector Selector: Select interrupt vector from location $(140 + X \times 4)_8$ , where X = Octal value of bits 32-34.
35	INTR-11	Interrupt PDP-11 if this bit is set.

The function SENSE STATUS sets the X-BUS to SENSE mode. An "IN" after setting BUS to this mode causes the buffer to be filled with identical words. Figure D.3 and Table D-3 explain the structure of right half of these words.

READ DATA function sets the BUS to a "READ" mode. An "IN" on BUS after setting it in this mode causes buffer to be filled with contents of successive words. However, if there are two or more buffers in input or output buffer rings it is difficult to determine which buffer was filled with required data. It is, therefore, suggested that the user explicitly request monitor to set up only single buffer rings using INBUF and OUTBUF UUOs.

Finally, as of writing this report, our DECsystem-10 monitor does not have the software required to handle interrupts from PDP-11, and so all function's involving interruption of PDP-10 (e.g. SENSE INTERRUPT) are inoperative.



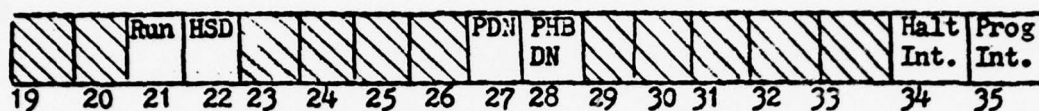


Figure D.3 : Bit assignment in PDP-11 Status word.

TABLE D-3 : Explanation of PDP-11 Status Word.

Bit	Mnemonic	Description
21	Run	PDP-11 is running.
22	HSD	Halt Switch Down : PDP-11's console halt switch is down.
27	PDN	PDP-11's Power is Down.
28	PHBDN	Power Has Been Down : PDP-11 power was down atleast once after status was sensed last time.
34	Halt Int.	Current interrupt to PDP-10 is due to PDP-11 coming to a halt.
35	Prog. Int.	Current interrupt to PDP-10 is due to PDP-11 program requesting it.

APPENDIX E

ON CONTROL THEORETIC APPROACH TO MEMORY MANAGEMENT

A B S T R A C T

Program page reference behavior can be modelled as a zero-one (binary) stochastic process. The process is non-stationary. A general ARIMA (m, d, n) model is proposed for this process. This model is then used to design an ARIMA-WS policy for memory management. In this policy the working set window size is dynamically adjusted to match the program locality size. Above all, the model provides a control theoretic derivation of Working Set policy. The limitations of the proposed model and possible directions for future research are also discussed.

## I. INTRODUCTION

The main problem in memory management is that of page replacement i.e., determining which page(s) should be kept in, or removed from, the memory. Obviously, a page which is going to be referenced in near future should be kept in the memory. The problem is, therefore, to predict the future page reference pattern from past observations. In recent years, Control theorists have done considerable progress on prediction and filtering theory. On the other hand, the flexibility and high speed of digital computers have resulted in the neglect of its dynamic performance optimization considerations. As a result most of the prediction schemes (page replacement algorithms) used in today's operating systems are primitive as compared to those available in control theory.

Even though the advantages of using modern control theory in computer systems are likely to be enormous, not much work has been done in this area. The first significant attempt in this direction is that of Charles Arnold [Arn75]. Arnold has shown that the page reference behaviour can be represented as a binary stochastic process and that Wiener filter theory can then be used to predict the future from the past behaviour of the process. Based on his Wiener predictor he has concluded that the working set (WS) policy is the optimal memory management policy. Arnold's

work has given us a new direction for memory management. However, in his initial attempt some very important points have been overlooked. This paper brings out these points, modifies the model as far as possible, and explains why the conclusions reached from the model need not be universally true.

The main result of the investigations reported here is the development of a so called "ARIMA-WS" policy of memory management wherein the WS ( Working Set ) window size is dynamically adjusted to match the program locality size. This not only is expected to result in a more efficient memory policy but also provides a control theoretic derivation of WS policy.

The organization of this paper is as follows. In section II we formulate the memory management problem as a prediction problem and briefly describe the stochastic process model of program behaviour proposed by Arnold. The discrepancies of this model are described in section III, and an ARIMA model is proposed as a remedy. Section IV exemplifies how the proposed model could be used to design a memory management policy. The limitation of our model and analysis method are discussed in section V. Finally, section 6 summarizes the main results.



## II. FORMULATION OF MEMORY MANAGEMENT AS A PREDICTION PROBLEM

### 2.1 Existing Prediction Algorithms for Replacement :

As said before, the problem of memory management is basically that of page replacement. Obviously, the best page to remove is the one that will never be needed again or, at least, not for a long time. In fact, it has been proved that for fixed memory partitioning, the best page to remove is the one that will not be referenced for the longest interval of time. This policy called 'MIN' is optimal in the sense that it minimizes the total number of page faults [Bel66]. However, this requires advance knowledge of the future page references (a prediction problem!). A realizable approximation to MIN is LRU policy which assumes that the page that has not been referenced for the longest interval in the past is the one that will not be referenced for the longest interval in future, and is the candidate for replacement.

In case of variable memory partitioning, it has been shown that MIN and its LRU approximations are not optimal. The optimal page replacement policy in this case (called VMIN algorithm) is to remove all those pages that will not be referenced during the next  $T$  time interval  $(t, t+T)$ , where  $T = R/U$  is the ratio of the cost of bringing a new page in the main memory from secondary memory to the cost of keeping a page in the main memory for unit time [PrF76].



Again, this is only of theoretical interest, because it requires knowledge of the future page reference string.

A realizable approximation to VMIN policy is the Working Set Policy [Den68]. According to this policy, the pages most likely to be referenced in the next  $T$  interval  $(t, t+T)$  are those which have been referenced during last  $T$  interval  $(t-T, t)$ . All other pages can therefore be removed. The interval  $T$  is called the window size.

## 2.2 Prediction Problem :

It is obvious from the above discussion that both LRU and WS try to predict future reference pattern from the past behaviour of the program. The prediction problem encountered here is similar to the prediction of stochastic processes encountered in other fields, e.g., prediction of stock prices. If we can somehow model the page reference string as a stochastic process, we can use modern control theoretic prediction algorithms such as Wiener filter, or Kalman filter etc. to predict future page reference string.

Ideally the model should be such that it incorporates all the information contained in the page reference string. However, such a model becomes very complex and difficult to analyze. We, therefore, choose to begin with a rather simple stochastic process model suggested by Arnold. More complexity may be introduced in future work. The implications of this simplification, and limitations of the

conclusion drawn from this model are discussed in the last section of this paper. It turns out that even this simplified model gives us much useful insight in to the problem. The stochastic process is therefore described next.

### 2.3 Stochastic Process Model :

The page reference pattern of a given ( say  $i$ th ) page of a program can be modelled as a zero-one process as follows :

$$p(k) = \begin{cases} 1 & \text{if the page is referenced in the} \\ & \text{kth interval ( } kT < t < (k+1)T \text{ )} \\ 0 & \text{otherwise} \end{cases}$$

A sample trajectory of the process is shown in figure 1. Without loss of generality we assume, hereafter, that the interval  $T$  is unity.

### 2.4 Wiener Filter Predictor :

As explained above, the problem of page replacement is that of predicting  $p(t+1)$  given trajectory up to time  $t$ , i.e., finding the best estimate  $\hat{p}(t+1)$  of  $p(t+1)$  from measurements up to time  $t$ . This problem is well known in control theory. There, much work has been done on the prediction of stochastic processes. A classic solution to the prediction problem is due to Wiener. It consists of designing a linear system (Wiener filter) with impulse response  $h(u)$  such that the output of the system is the estimate  $\hat{p}(t+1)$  when input is  $p(k)$ ,  $0 < k < t$  ( see figure

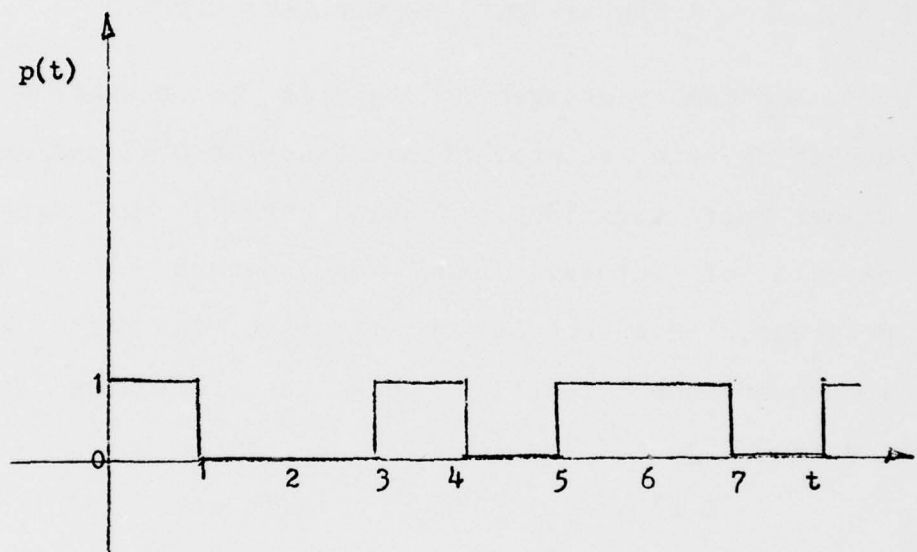


Figure 1 : References to a page modelled as a binary stochastic process

2 ). The impulse response  $h(u)$  can be obtained by solving the Wiener-Hopf equation :

$$C(k+1) = \sum_u C(k-u)h(u) \quad k=0, 1, 2, \dots$$

where  $C(k)$  = autocovariance function of  $p(t)$

$$= E[p(t+k)p(t)] - E[p(t+k)]E[p(t)]$$

The memory management problem is therefore that of measuring the autocovariance function  $C(k)$  and solving the Wiener-Hopf equation. Arnold [Arn75] has reported the results of autocovariance measurements on a number of programs. His conclusion is that in most cases the autocovariance function has the following form ( see figure 3 ) :

$$C(k) = a + jb^k \quad \text{with } a > 0 \text{ and } j=\text{constant}$$

Using this formula for  $C(k)$ , Arnold has solved the Wiener-Hopf equation, devised a memory management policy, and drew several conclusions. However, we do not agree with Arnold's solution and his conclusions for reasons discussed in the next section.

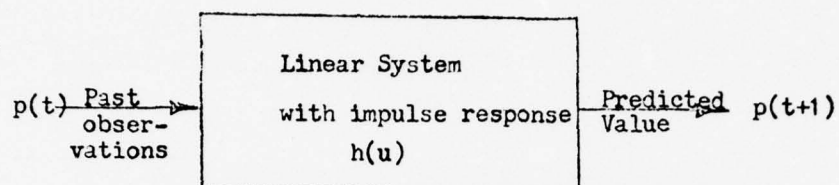


Figure 2 : Wiener filter predictor.  $h(u)$  is given by solution to Wiener-Hopf equation.

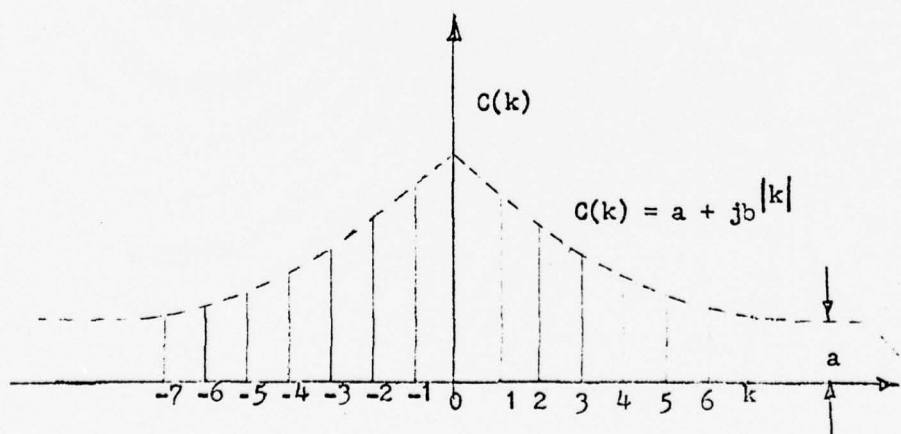


Figure 3 : Observed autocovariance function of page reference process  $p(t)$ . [Arnold75]



III. THE ARIMA MODEL OF PROGRAM BEHAVIOUR3.1 Objection to Arnold's Solution :

The main objection to Arnold's solution is that the process  $p(t)$  is non-stationary. A test for stationarity, commonly used in time series analysis [Boj70] is to verify that the autocovariance function  $C(k)$  dies down to zero as the lag  $k$  tends to infinity. Arnold reports in his paper that one of the main findings of his measurements was the fact that the autocovariance function  $C(k)$  does not go to zero, i.e., the constant  $a \neq 0$ . However, he failed to notice that this also implies the non-stationarity.

For a non-stationary process the covariance of  $p(t_1)$  and  $p(t_2)$  is a function  $C(t_1, t_2)$  of two variables  $t_1, t_2$  rather than the difference  $t_1 - t_2$ . The autocovariance function  $C(k)$  does, therefore, not have any meaning, and correlation techniques like Wiener filter can not be used.

In fact, it can be easily verified by substituting Arnold's solution in Wiener-Hopf equation, that the equation is satisfied if and only if  $a=0$ , i.e., when the process is stationary.

3.2 Non-stationary Model :

There are unlimited number of ways in which a process can be non-stationary. However, most of the real world non-stationary processes exhibit a homogeneous

non-stationary behaviour such that some suitable difference of the process is stationary. For example, if the process exhibits homogeneity in the sense that apart from local level ( i.e., local mean ), one part of the series behaves much like any other part, then the first difference of the process may be found to be stationary.

To model such non-stationary processes, therefore, one studies the autocovariance function of 1st, 2nd, 3rd, ... differences until a stationary process is obtained. Thus, to model  $p(t)$  we should study the autocovariance functions of

$$Dp(t) = p(t+1) - p(t)$$

$$D^2p(t) = Dp(t+1) - Dp(t)$$

...

...

$$D^d p(t) = D^{d-1} p(t+1) - D^{d-1} p(t)$$

till a stationary process  $D^d p(t)$  is found.

The non-stationarity of page reference process  $p(t)$  can be explained as follows. Even though the program behaviour may be stationary in one locality, the frequency of reference to a particular page varies as the program progresses from one locality to the next. Thus, the process  $p(t)$  may behave like a set of locally stationary processes, i.e., like a homogeneous non-stationary process whose mean value varies. If this is so, the first difference of  $p(t)$

must be stationary. This is just a hypothesis. Actual calculations are required to confirm this. We have not yet done such calculations. However, regardless of which difference of  $p(t)$  turns out to be stationary, there are some interesting conclusions that would always be valid.

Suppose that the  $d$ th difference of  $p(t)$  comes out to be stationary, then  $p(t)$  can be modelled as an Autoregressive Integrated Moving average process model of order  $m, d, n$ , i.e., ARIMA( $m, d, n$ ) model :

$$\begin{aligned} q(t+m) + a_1 q(t+m-1) + a_2 q(t+m-2) + \dots + a_m q(t) \\ = b_0 e(t+m-1) + b_1 e(t+m-2) + \dots + b_n e(t+m-n-1) \end{aligned}$$

where  $q(t) = D^d p(t)$ , and  $e(t)$  is white noise.

The procedure to determine orders  $m, d, n$  and coefficient  $a_i, b_j$  are described in Box and Jenkins [BoJ70].

#### IV. MEMORY MANAGEMENT USING ARIMA(m,d,n) MODEL - AN EXAMPLE

In this section we discuss how the ARIMA(m, d, n) model could be used to design a simple memory management technique, which is expected to perform better than existing techniques. In the following discussion we assume that the process  $p(t)$  has been found to be an ARIMA(1, 1, 0) process. however, this is just an example. Analysis for any other order model would result in similar results.

##### 4.1 ARIMA(1,1,0) Model :

Let  $q(t)$  represent the first difference of  $p(t)$ , i.e.,

$$q(t) = Dp(t) = p(t+1) - p(t) \quad (4.1.1)$$

Suppose that we discover that the autocorrelation function of  $q(t)$  is an exponential as shown in figure 4a, or a exponentially damped sinusoid as shown in figure 4b. In either case, we can fit a first order autoregressive model to  $q(t)$  i.e.,

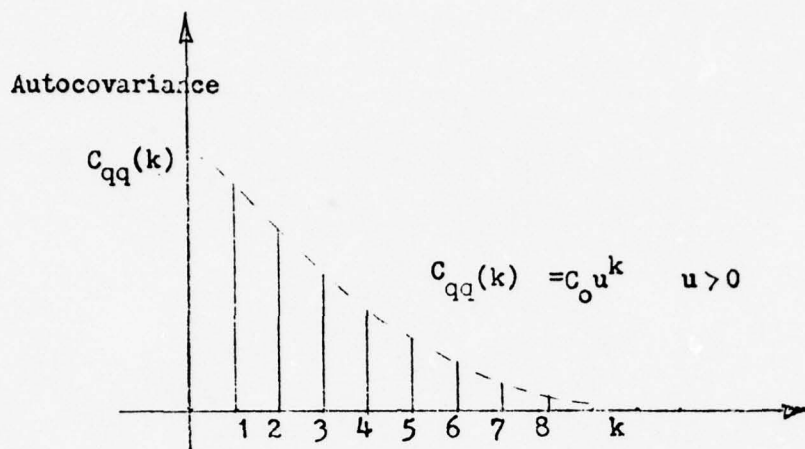
$$q(t+1) = aq(t) + be(t)$$

where  $a$  and  $b$  are parameters to be determined, and  $e(t)$  is unit variance white noise. The process  $p(t)$  can then be represented by a two variable state equation as follows :

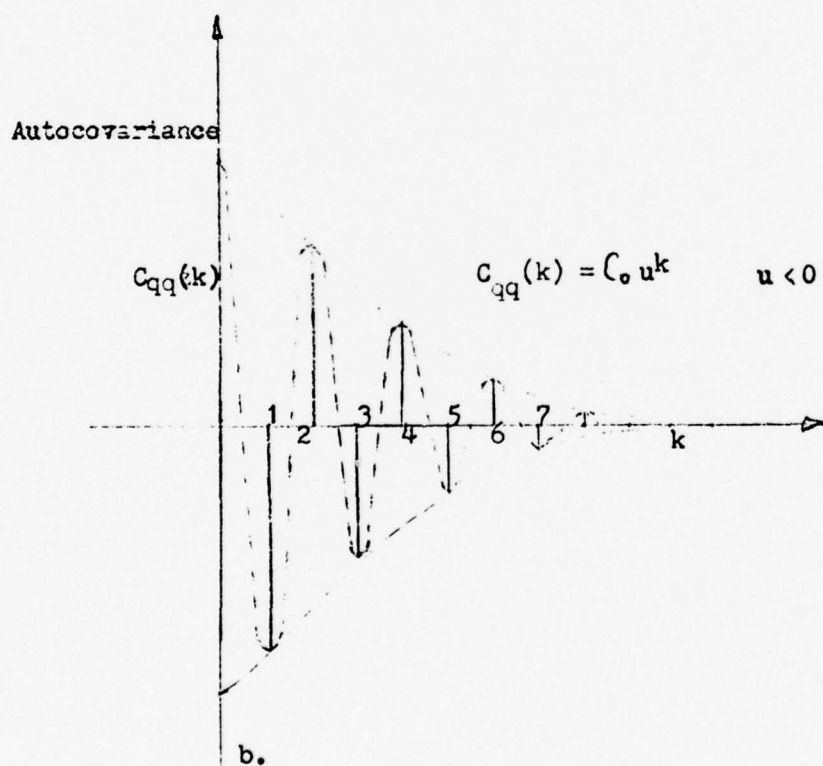
$$\begin{array}{rcl} p(t+1) & 1 & 1 \quad p(t) \\ q(t+1) & = & 0 \quad a \quad q(t) \end{array} + \begin{array}{c} 0 \\ b \end{array} e(t) \quad (4.1.2)$$

$$z(t) = \begin{array}{c} 1 \quad 0 \\ \quad \quad \quad \end{array} \begin{array}{c} p(t) \\ q(t) \end{array}$$

where  $z(t) = p(t)$  is the output of a two dimensional linear



a.



b.

Fig. 4 : Two possible forms of autocovariance of first difference of  $p(t)$ . If  $C_{qq}(t)$  is exponential as in a, or exponentially damped sinusoid as in b,  $q(t)$  can be modelled as a first order autoregressive process.



system driven by  $e(t)$  as shown in figure 5. This model of  $p(t)$  could also be called "Kalman Filter Representation" of page reference behaviour. Notice, however, that in this case there is no measurement noise. There are many techniques for estimating the parameters  $a$  and  $b$ . One simple technique is to solve Yule-Walker equations. The solution in this case would be :

$$\begin{aligned} a &= \frac{C_{qq}(1)}{C_{qq}(0)} \\ b &= C_{qq}(0)[1-a^2] \end{aligned} \quad (4.1.3)$$

Where  $C_{qq}(k)$  is the autocovariance function of  $q(t)$  at lag  $k$ .

#### 4.2 Memory Management Using ARIMA(1,1,0) model :

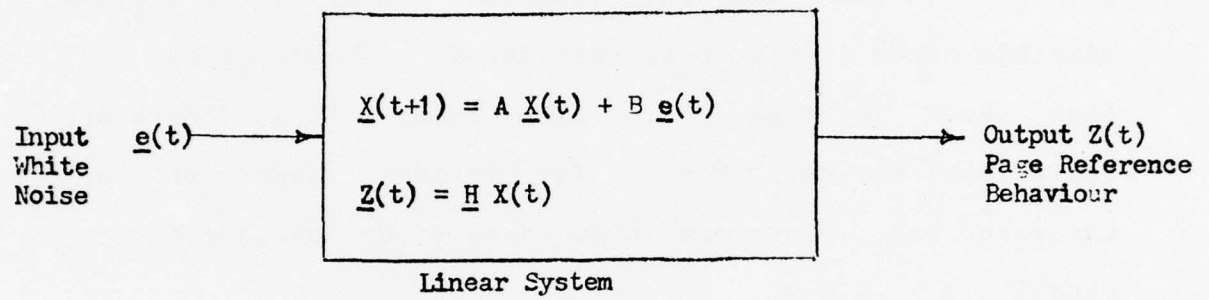
After eliminating  $q(t)$  from equation 4.1.2 we get the following equation in  $p(t)$  :

$$p(t+1) = (1+a)p(t) - ap(t-1) + be(t) \quad (4.2.1)$$

Where  $-1 < a < 1$  for stationarity of  $q(t)$ . Since  $e(t)$  is white noise term, it can not be predicted and the best estimate  $\hat{p}(t+1)$  is given by:

$$\hat{p}(t+1) = (1+a)p(t) - ap(t-1) \quad (4.2.2)$$

One way to design a page replacement algorithm based on this equation is to choose a cut off point say at 0.5 and to keep a page if the predicted value  $\hat{p}(t+1) \geq 0.5$  and to remove it otherwise. If keeping a page is denoted by boolean variable  $\check{p}(t+1)$  then the above management policy



$$\underline{A} = \begin{bmatrix} 1 & 0 \\ 0 & a \end{bmatrix}$$

$$\underline{B} = \begin{bmatrix} 0 \\ b \end{bmatrix}$$

$$\underline{H} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

Figure 5 : State variable representation of Program behaviour.

is given by

$$\hat{p}(t+1) \geq 0.5 \Rightarrow \bar{p}(t+1)=1 \text{ (keep)}$$

$$\hat{p}(t+1) < 0.5 \Rightarrow \bar{p}(t+1)=0 \text{ (remove)}$$

where  $\hat{p}(t+1)$  is given by equation 4.2.2. Since both  $p(t)$  and  $p(t-1)$  can take only two values 0 or 1, there are 4 possible cases in all to be considered. These cases have been shown in figure 6. Also shown in that figure are replacement decisions  $\bar{p}(t+1)$  for various ranges of the parameter "a". There are three cases to be considered.

Case I :  $a > -0.5$

As shown in figure 6, in this case  $\bar{p}(t+1) = p(t)$ , i.e., keep only those pages that were referenced in the last T (unit) interval. This is exactly what Working Set (WS) policy also says. Thus for  $a > -0.5$ , the ARIMA(1,1,0) policy is the same as WS policy with unit window size.

Case II :  $a = -0.5$

Again from figure 6, we see that for  $a = -0.5$   $\bar{p}(t+1) = p(t) \vee p(t-1)$ \* i.e., all pages that were referenced in the last two interval should be kept in the memory. This is equivalent to using a Working Set policy with window size of 2.

Case III :  $a < -0.5$

In this case  $\bar{p} = p(t-1)$  i.e., keep the pages that were

---

\* Here  $\vee$  represents logical OR operation on boolean variables.

<u>p(t-1)</u>	<u>p(t)</u>	<u><math>\hat{p}(t+1)</math></u>	<u><math>\tilde{p}(t+1)</math></u>		
			<u>a &gt; -.5</u>	<u>a = -.5</u>	<u>a &lt; -.5</u>
0	0	0	0	0	0
0	1	1+a	1	1	0
1	0	-a	0	1	1
1	1	1	1	1	1

Figure 6. Prediction using ARIMA(1,1,0) model.

referenced in the last but one interval. This happens if the program has a big loop (or locality) as shown in figure 7. In this case, consider the instant when the program is in the center of the loop. Obviously, the set of pages to be referenced in the next interval  $(t, t+1)$  is not the set that was referenced in the last interval  $(t-1, t)$ , but the set referenced in the last but one interval  $(t-2, t-1)$ . In general, when locality size is larger than the window size, this situation would be obtained.

The parameter "a" in ARIMA(1,1,0) model acts as a measure of difference in the WS window size and actual program locality size. This property of "a" can be used to design a modified WS policy where window size is adaptively changed to match the program locality size using "a" as an indicator. Hereafter, this policy will be called ARIMA-WS policy. It can be formally stated as follows :

ARIMA-WS POLICY : Start with a WS policy with a small window size. Increase the size (say double) if the parameter "a" is found to be less than or equal to -0.5 .

In the WS policy, as it is currently used, the window size is arbitrarily chosen. The choice of proper window size is a difficult decision because whatever value is chosen, there will exist a set of programs for which the chosen size is too small, and another set for which the chosen size is very large. The adaptive choice of window



```

DO 9000 I=1,1000
-----
CALL SUBR1
-----
CALL SUBR2
-----
9000 CONTINUE

```

- a. A program with big locality size.  
Its expanded version is shown below.

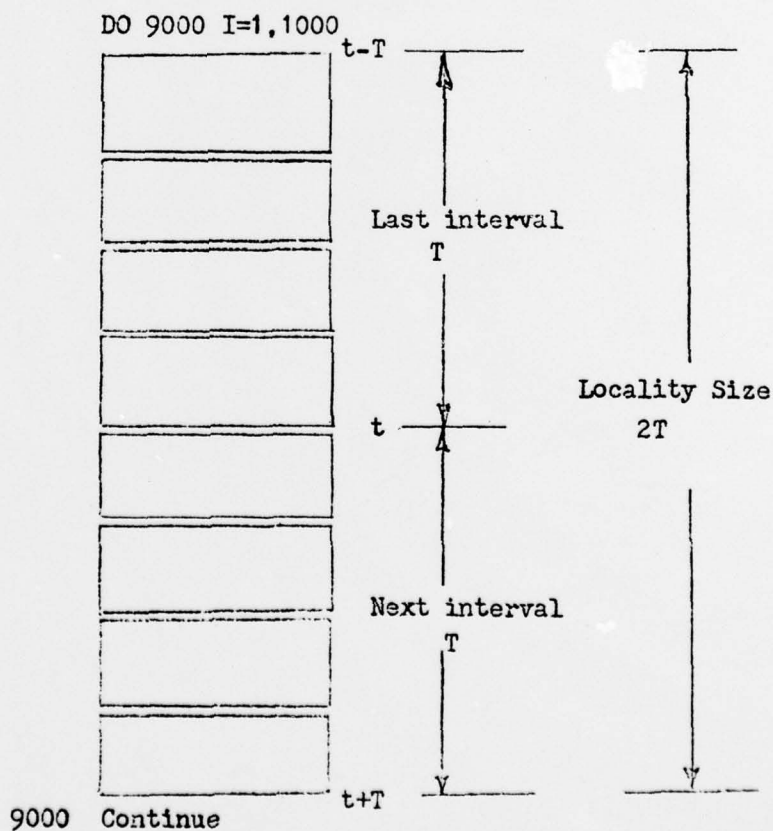


Fig. 7 : An example where locality size is larger than window size. The set of pages to be referenced in the next interval is that of pages referenced in last but one interval.

size in ARIMA-WS using "a" as a guide is, therefore, a definite improvement over ordinary WS policy.

At this point let us reiterate that a suitable ARIMA-WS policy can be derived regardless of the order of the model. For any general ARIMA(m,d,n) policy, one can always divide the parameter space into subsets so that if the parameters are evaluated to lie in one subset it can be taken as an indication to the fact that window size is smaller than the locality size. Similarly, the parameters would lie in the other subset if the window size is too large.

size in ARIMA-WS using "a" as a guide is, therefore, a definite improvement over ordinary WS policy.

At this point let us reiterate that a suitable ARIMA-WS policy can be derived regardless of the order of the model. For any general ARIMA(m,d,n) policy, one can always divide the parameter space into subsets so that if the parameters are evaluated to lie in one subset it can be taken as an indication to the fact that window size is smaller than the locality size. Similarly, the parameters would lie in the other subset if the window size is too large.

## V. LIMITATIONS OF THE ARIMA MODEL

In the last section we have shown that the best prediction scheme using the ARIMA model is a WS policy provided the window size is properly adjusted. However, this should not be interpreted to mean that WS with proper window size is "the optimal" page replacement algorithm. WS may probably be the optimum for the class of models describable by the zero-one stochastic process model that we started with. Unfortunately, this model uses only a subset of the available past information. It is quite possible to design a better policy if we start with a model having more information e.g., using the number of times a page is referenced in any interval. The ideal model would be one that uses the complete past information, i.e., the complete page reference string.

### 5.1 Incomplete Information :

A deeper insight into the information structure can be obtained by considering the information used by various algorithms. VMIN - the optimal variable space algorithm uses the complete page reference string. WS requires knowledge of set of pages referenced in the last T interval. It does not require the order in which the pages were referenced or the number of times they were referenced. A general ARIMA model would use all the sets of pages referenced in successive T interval. Finally, LRU uses the

Page 62

set of last referenced  $m$  pages along with their order of reference. The Venn diagram of information used by these algorithms is shown in figure 8. The broken line in this figure separates the past from the future. There are two inferences to be drawn from this figure :

1. The information used by WS is a subset of that used by ARIMA. This explains why the ARIMA policy can always be specialized to a WS set policy with proper window size.
2. There is much information, ( like the frequency of reference of a page in an interval, and the order of reference of various pages ), that is not used in the zero-one stochastic process model used to derive ARIMA policy. In future, we intend to develop a model which uses complete past information so that both WS and LRU will be special cases of this generalized model. The conclusions drawn would then be universal in scope.

## 5.2 Approximation introduced by Gaussian Analysis :

So far, we have analyzed the zero-one stochastic process as if it were a normal process. For example, we assume it to be the output of a linear system driven by gaussian white noise. The analysis is, therefore, approximate. The reason for this choice of method for initial analysis is that there is no convenient way of modelling non-gaussian processes. A more accurate analysis would be obtained by taking the binary nature of the process into account. We are currently working on developing modelling and identification



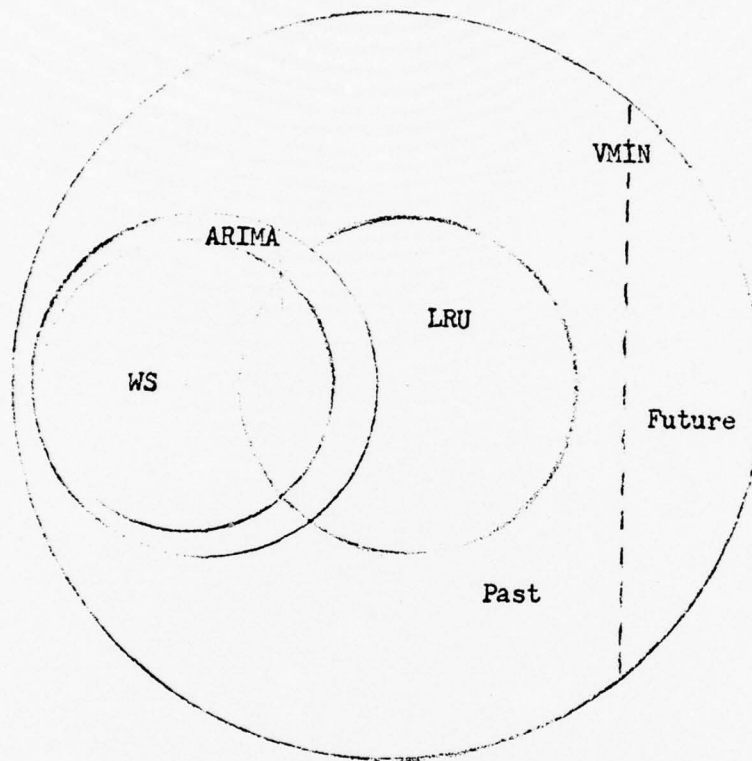


Fig. 8 : Information used by various page replacement algorithms.

techniques for binary process using boolean switching  
circuits driven by binary equivalent of white noise.

## VI. CONCLUSION

The page reference behaviour modelled as a zero-one binary stochastic process exhibits a non-stationary behaviour. Therefore, the Wiener filter predictor proposed by Arnold can not be used. A general ARIMA(m, d, n) model should be used for the process. This model can then be used to design a memory management policy.

The main results achieved so far can be stated as follows:

1. We have a control theoretic model of program page reference behaviour.
2. We have been able to design an algorithm to dynamically adjust the WS window size.
3. We have a control theoretic derivation of WS policy.

The zero-one stochastic model does not use all the available information and further approximations are introduced by gaussian analysis. We, therefore, expect that the development of a stochastic process model that uses all available information and development of identification methods for discrete binary processes will lead to better understanding and management of program memory behaviour.

# REFERENCES

- [Arn75] Arnold, C. R., "A Control Theoretic Approach to Memory Management", Proceedings Ninth Asilmor Conference on Circuits, Systems, and Computer, Pacific Grove, Calif., November 1975.
- [Bel66] Belady, L. A., "A Study of Replacement Algorithms for a Virtual Storage Computer", IBM System Journal, Vol. 5, No. 2, 1966, pp. 78-101.
- [BoJ70] Box, G. E. P., and Jenkins, G. M., "Time Series Analysis Forecasting and Control", Holden-Day, 1970.
- [Den68] Denning, P. J., "The Working Set Model for Program Behaviour", CACM, Vol. 11, No. 5, May 1968, pp. 323-333.
- [PrF76] Prieve, B. G., and Fabry, R. S., "VMIN - An Optimal Variable Space Page Replacement Algorithm", CACM, Vol. 19, No. 5, May 1976, pp. 295-297.

# METRIC SYSTEM

## BASE UNITS:

Quantity	Unit	SI Symbol	Formula
length	metre	m	...
mass	kilogram	kg	...
time	second	s	...
electric current	ampere	A	...
thermodynamic temperature	kelvin	K	...
amount of substance	mole	mol	...
luminous intensity	candela	cd	...

## SUPPLEMENTARY UNITS:

plane angle	radian	rad	...
solid angle	steradian	sr	...

## DERIVED UNITS:

Acceleration	metre per second squared	...	m/s
activity (of a radioactive source)	disintegration per second	...	(disintegration)/s
angular acceleration	radian per second squared	...	rad/s
angular velocity	radian per second	...	rad/s
area	square metre	...	m
density	kilogram per cubic metre	...	kg/m
electric capacitance	farad	F	A·s/V
electrical conductance	siemens	S	A/V
electric field strength	volt per metre	...	V/m
electric inductance	henry	H	V·s/A
electric potential difference	volt	V	W/A
electric resistance	ohm	...	V/A
electromotive force	volt	V	W/A
energy	joule	J	N·m
entropy	joule per kelvin	...	J/K
force	newton	N	kg·m/s
frequency	hertz	Hz	(cycle)/s
illuminance	lux	lx	lm/m
luminance	candela per square metre	...	cd/m
luminous flux	lumen	lm	cd·sr
magnetic field strength	ampere per metre	...	A/m
magnetic flux	weber	Wb	V·s
magnetic flux density	tesla	T	Wb/m
magnetomotive force	ampere	A	...
power	watt	W	J/s
pressure	pascal	Pa	N/m
quantity of electricity	coulomb	C	A·s
quantity of heat	joule	J	N·m
radiant intensity	watt per steradian	...	W/sr
specific heat	joule per kilogram-kelvin	...	J/kg·K
stress	pascal	Pa	N/m
thermal conductivity	watt per metre-kelvin	...	W/m·K
velocity	metre per second	...	m/s
viscosity, dynamic	pascal-second	...	Pa·s
viscosity, kinematic	square metre per second	...	m/s
voltage	volt	V	W/A
volume	cubic metre	...	m
wavenumber	reciprocal metre	...	(wave)/m
work	joule	J	N·m

## SI PREFIXES:

Multiplication Factors	Prefix	SI Symbol
1 000 000 000 000 = 10 <sup>12</sup>	tera	T
1 000 000 000 = 10 <sup>9</sup>	giga	G
1 000 000 = 10 <sup>6</sup>	mega	M
1 000 = 10 <sup>3</sup>	kilo	k
100 = 10 <sup>2</sup>	hecto*	h
10 = 10 <sup>1</sup>	deka*	da
0.1 = 10 <sup>-1</sup>	deci*	d
0.01 = 10 <sup>-2</sup>	centi*	c
0.001 = 10 <sup>-3</sup>	milli	m
0.000 001 = 10 <sup>-6</sup>	micro	μ
0.000 000 001 = 10 <sup>-9</sup>	nano	n
0.000 000 000 001 = 10 <sup>-12</sup>	pico	p
0.000 000 000 000 001 = 10 <sup>-15</sup>	femto	f
0.000 000 000 000 000 001 = 10 <sup>-18</sup>	atto	a

\* To be avoided where possible.



*MISSION  
of  
Rome Air Development Center*

*RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C<sup>3</sup>) activities, and in the C<sup>3</sup> areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.*

